Ph.D. DISSERTATION

TIMING ANALYSIS AND OPTIMIZATION IN LOGIC AND PHYSICAL SYNTHESIS

로직 및 피지컬 합성에서의 타이밍 분석과 최적화

BY

JEONGWOO HEO

AUGUST 2020

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

TIMING ANALYSIS AND OPTIMIZATION IN LOGIC AND PHYSICAL SYNTHESIS

로직 및 피지컬 합성에서의 타이밍 분석과 최적화

BY

JEONGWOO HEO

AUGUST 2020

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

TIMING ANALYSIS AND OPTIMIZATION IN LOGIC AND PHYSICAL SYNTHESIS

로직 및 피지컬 합성에서의 타이밍 분석과 최적화

지도교수 김 태 환
이 논문을 공학박사 학위논문으로 제출함

2020년 6월

서울대학교 대학원

전기·정보 공학부

허 정 우

허정우의 공학박사 학위 논문을 인준함

2020년 6월

위 원 장: _____
부위원장: _____
위 원: _____
위 원: _____
위 원: _____

Abstract

Timing analysis is one of the necessary steps in the development of a semiconductor circuit. In addition, it is increasingly important in the advanced process technologies due to various factors, including the increase of process–voltage–temperature variation. This dissertation addresses three problems related to timing analysis and optimization in logic and physical synthesis. Firstly, most static timing analysis today are based on conventional fixed flip-flop timing models, in which every flip-flop is assumed to have a fixed clock-to-Q delay. However, setup and hold skews affect the clock-to-Q delay in reality. In this dissertation, I propose a mathematical formulation to solve the problem and apply it to the clock skew scheduling problems as well as to the analysis of a given circuit, with a scalable speedup technique. Secondly, near-threshold computing is one of the promising concepts for energy-efficient operation of VLSI systems, but wide performance variation and nonlinearity to process variations block the proliferation. To cope with this, I propose a holistic hardware performance monitoring methodology for accurate timing prediction in a near-threshold voltage regime and advanced process technology. Lastly, an asynchronous circuit is one of the alternatives to the conventional synchronous style, and asynchronous pipeline circuit especially attractive because of its small design effort. This dissertation addresses the synthesis problem of lightening two-phase bundled-data asynchronous pipeline controllers, in which delay buffers are essential for guaranteeing the correct handshaking operation but incurs considerable area increase.

keywords: Timing analysis, flip-flop, hardware performance monitoring methodology, near-threshold computing, asynchronous circuit, pipeline controller, delay path.
student number: 2014-21722

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Flexible Flip-Flop Timing Model

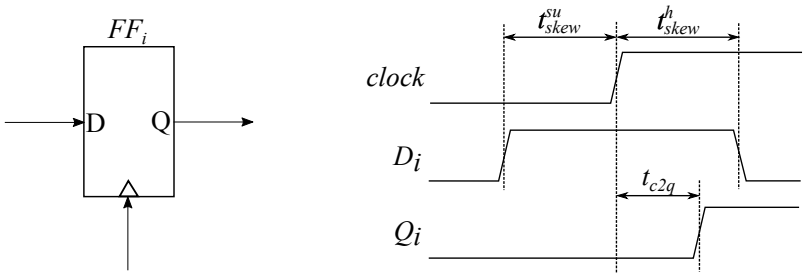
Almost all static timing analysis (STA) techniques used today for evaluating timing performance of sequential circuits are based on setup and hold times constrained flip-flop timing model. This timing model entails that a clock-to-Q delay of a flip-flop is set to a constant value and the setup and hold times are measured at the points of 5%$\sim$10% clock-to-Q delay increase with additional timing margin for variations. However, it has been regarded that a clock-to-Q delay has nothing to do with the setup and hold skews of the flip-flop; consequently, the timing model simply sets the setup and hold times to be unnecessarily large in order to overly constrain timing at flip-flops, so that the actual clock-to-Q delay is always shorter than the (assumed) maximum clock-to-Q delay. Thus, this *fixed flip-flop timing model* is safe in that if there is no timing violation, it ensures that the circuit has really no setup and hold time violation at all. However, due to the pessimistic setup and hold time constraints independently imposed on flip-flops, the timing reports may not be so accurate.

Figure 1.1 shows SPICE simulation results for the change of the values of a setup skew, a hold skew, and a clock-to-Q delay. Illustration for these notions is shown in Figure 1.1(a). The three curves in Figure 1.1(b) show the trade-off between setup and

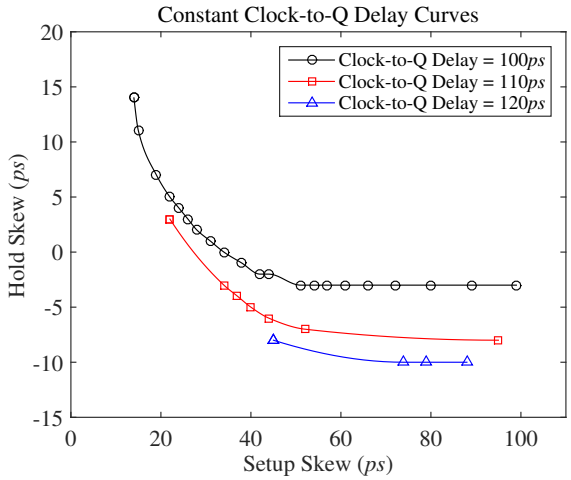
hold skews for the clock-to-Q delays of 100ps, 110ps, and 120ps, respectively. These curves clearly reveal that timing analysis and optimization should take into account the interdependent relation between a setup skew, a hold skew, and a clock-to-Q delay, if designers want to avoid unnecessary late-stage engineering change order (ECO) or high cost design change before timing sign-off due to inaccurate analysis of the timing behavior of a circuit. In this dissertation, I refer to this property as *flexible flip-flop timing* to differentiate it from the concept of the conventional fixed flip-flop timing.

In the past, many researchers studied about characterization, analysis, and optimization methods of the flexible flip-flop timing model widely. Stojanovic and Oklobdzija [11] focused on the analysis of the relationship between a setup skew and a clock-to-Q delay, and minimized the sum of them. Jain and Blaauw [12] modeled the relation between a clock-to-Q delay and a setup skew in the form of an exponential expression to improve the analysis accuracy over that in [11]. Both of the works, however, did not consider the effect of a hold skew on a clock-to-Q delay. To overcome this shortcoming, Rao and Howick [13] firstly clarified the existence of the close interaction between a setup skew, a hold skew, and a clock-to-Q delay. Srivastava and Roychowdhury [14, 15] developed a methodology of fast characterization of the timing relation by employing Newton-Raphson solution and Eular curve tracking of state-transition equations. Salman *et al.* [16, 17] enhanced the accuracy of timing analysis by reflecting the interdependent relation of setup and hold times and a clock-to-Q delay. Later, Salman and Friedman [18] attempted to utilize the interdependent timing relation to tolerate delay variations, so that delay uncertainty be reduced. While those works focused on STA, Hatami, Abrishami, and Pedram [19] demonstrated, through Monte-Carlo simulation, that the flexible flip-flop timing model can significantly improve the accuracy in statistical static timing analysis (SSTA).

In the perspective of analyzing and optimizing circuit timing, Chen, Li, and Schlichtmann [20] addressed the clock period minimization problem and proposed a simple and accurate analytical model of a clock-to-Q delay surface. They also expressed an



(a) The values of the setup skew t^{su}_{skew} and the hold skew t^{h}_{skew} of FF_i determine the value of the clock-to-Q delay t_{c2q} of FF_i. t_{c2q} would affect the values of t^{su}_{skew} and t^{h}_{skew} of the flip-flops to be driven by FF_i.



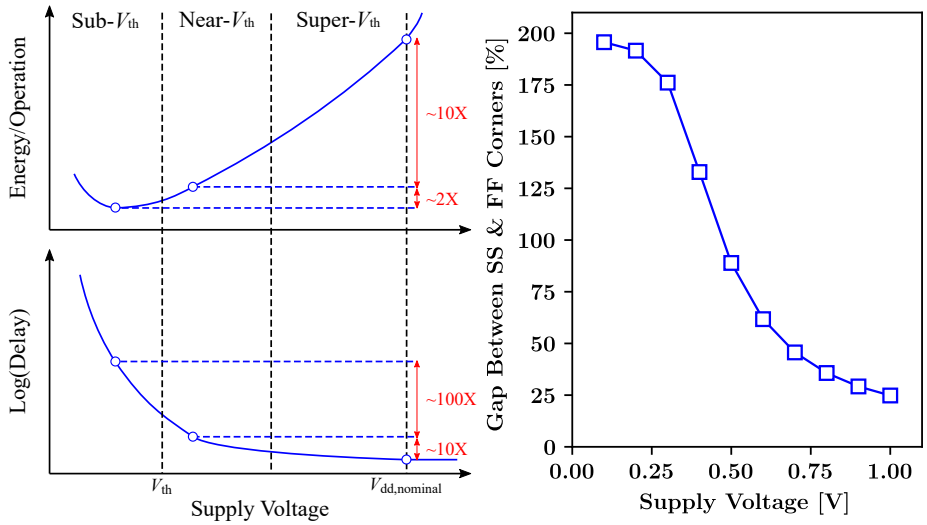
(b) Curves showing the relationship between a setup skew and a hold skew for the clock-to-Q delays of 100ps, 110ps, and 120ps. The flexibility of trading the setup skew with the hold skew increases as the clock-to-Q delay decreases.

Figure 1.1: Illustration for the interdependent relation between a setup skew, a hold skew, and a clock-to-Q delay. The curves were produced by SPICE simulations with 45nm NanGate Open Cell Library [4].

iterative timing analysis method, in which the clock-to-Q delay of a flip-flop is iteratively computed, starting from a valid clock-to-Q delay value, and reports a timing violation on a flip-flop if the clock-to-Q delay value of the flip-flop does not converge. One critical limitation is that if the initial delay value is improperly set, it may lead to a false negative result. They tried to find the minimum clock period of a circuit by repeatedly applying this analysis technique in a binary search framework. Kahng and Lee [21] divided the flexible flip-flop timing model based timing margin recovery problem into two optimization subproblems: (1) optimization between a setup time and a clock-to-Q delay and (2) optimization between a hold time and a clock-to-Q delay to enable linear programming formulations. However, the linearization of flexible flip-flop timing model causes non-trivial fitting errors. On the other hand, Seo, Heo, and Kim [22, 2] considered the flexible flip-flop timing in the useful clock skew scheduling problem. They proposed a stepwise clock skew scheduling technique, in which at each iteration, setup and hold slacks are systematically and incrementally relaxed by utilizing flexible flip-flop timing based analysis method in [20]. Though the approach is practical and fast, the result could be far from the optimal solution due to its ad hoc nature. Recently, Yang, Tam, and Jiang [23] attempted to compute timing relation accurately only on the timing critical part of a circuit.

1.2 Hardware Performance Monitoring Methodology

Near-threshold computing (NTC) [5] is one of the representative low power techniques that can break through the challenge of so-called dark silicon [24, 25] for IoT and newly emerging energy-efficient system designs such as artificial intelligence [26]. While reduced supply voltage (V_{dd}) deteriorates circuit performance, it saves power consumption considerably at the same time, as shown in Figure 1.2(a). Based on this property, NTC aims to improve the energy-efficiency of a circuit significantly by lowering down V_{dd} from a super-threshold voltage (super-V_{th}) regime ($V_{\mathrm{dd}} \gg V_{\mathrm{th}}$) to near-

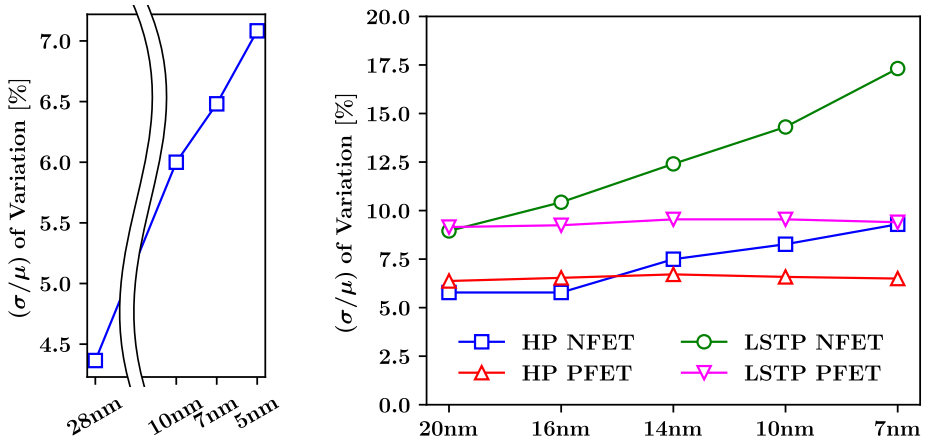


(a) (b)

Figure 1.2: (a) Energy per operation and delay in different V_{dd} regimes [5]. (b) Impact of V_{dd} scaling on the period of an inverter-based ring oscillator at 28nm process technology.

threshold voltage (NTV) regime ($V_{dd} \gtrsim V_{th}$) aggressively. Although the optimal point in terms of energy per operation is formed in a sub-threshold voltage (sub-V_{th}) regime ($V_{dd} < V_{th}$), it is hard to be the mainstream except for silicon products in some specific markets due to severe performance loss. By contrast, since energy-efficiency can be improved by about $10\times$ at the cost of about $10\times$ performance degradation in an NTV regime compared to super-V_{th} operation, NTC could be a more practical low-power operation than sub-V_{th} operation.

However, one of the barriers to the use of NTC is a wide gap in performance variation. For example, Figure 1.2(b) shows that difference between simulation results of SS and FF corners increases from 25% to 50% as V_{dd} decreases from 1.0V (super-V_{th} regime) to 0.6V (NTV regime). In cases of advanced process technology, it is more severe due to process variations (PVs). Figure 1.3(a) represents the variation of the effective channel length (L_{eff}) of a transistor for different process technologies, from



(a) (b)

Figure 1.3: (a) Variation of L_{eff} at 28nm, 10nm, 7nm, and 5nm process technologies taken from 28nm industry PDK and IRDS 2017 roadmap [6]. (b) Impact of gate workfunction variation on I_{on} for High Performance (HP) and Low Standby Power (LSTP) version of sub-22nm FinFET PTM-MG models [7]. Note that the y-axis represents the ratio between the standard deviation of a parameter variation and the average of it.

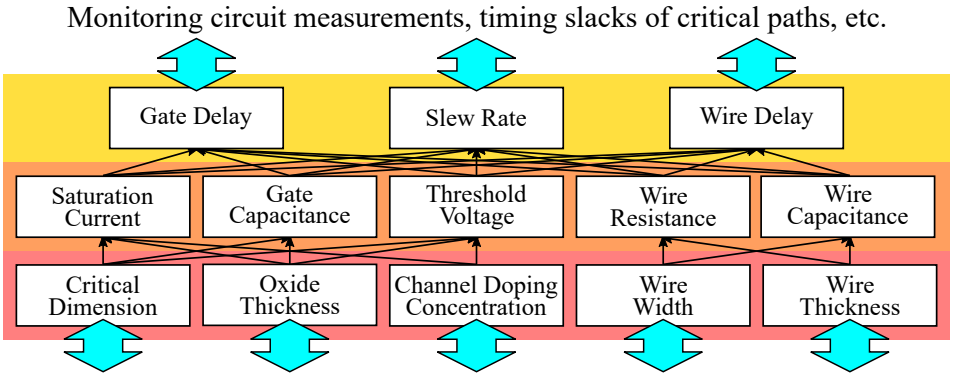
which I can see that it enlarges from 4.4% in a 28nm process to 6.0%, 6.5%, and 7.1% in 10nm, 7nm, and 5nm processes, respectively. In addition, Meinhardt, Zimpeck, and Reis [7] reported that the impact of the variation of gate workfunction from metal granularity on change of transistor on-current (I_{on}) increases by up to 2× in a 7nm process in comparison with that of a 20nm process, as shown in Figure 1.3(b). Hence, if I handle the problem by simply adding some margins as engineers did conventionally, the performance loss will be significant, which might lower the energy-efficiency of NTC than that of super-V_{th} operation.

The main reason is related to PVs [27, 28]. According to Blaauw, Chopra, Srivastava, and Scheffer [29], the limitation of manufacturing technology (e.g., chemical mechanical polishing) makes variations of physical parameters (e.g., critical dimension), electrical parameters (e.g., gate capacitance), and circuit timing (e.g., gate de-

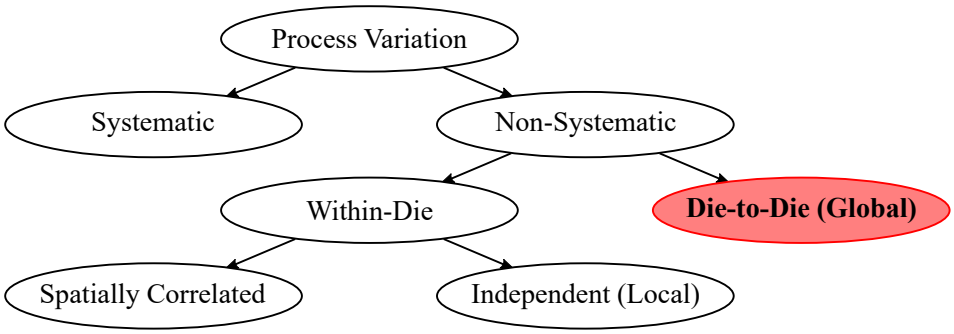
lay), sequentially, and leads to change of the whole VLSI system performance at last, as shown in Figure 1.4(a). They also classified PVs based on whether they are deterministic or statistical and the spatial scale over which they operate, as shown in Figure 1.4(b), and among them, die-to-die variations, or also referred to as global random variations, affect all the devices on the same die in the same way. *Hardware performance monitoring (HPM) methodology* exploits this property of global random variations for identifying the performance status of a target circuit or system through measuring monitoring circuits. It can also be used in conjunction with some post-silicon techniques such as adaptive voltage scaling (AVS) and speed binning.

Many researchers have proposed various kinds of monitoring circuits with a small area and test cost over the past years. For example, generic circuits such as process-sensitive ring oscillators [30, 31] and phase-locked loops [32] enable post-silicon adaptations straightforwardly despite its inaccuracy. On the contrary, some researchers (e.g., [33, 34]) suggested designing monitoring circuits by reflecting characteristics of a target circuit. Parametric monitors (e.g., [35, 36, 37]) can also contribute more accurate delay estimation for a design-specific delay model by coupling them with some specific process parameters with high sensitivities. Another group of the circuits called in-situ monitors (e.g., [38, 39, 40, 41, 42, 43, 44]) measures delays of some critical paths directly, but it has representativeness and area overhead issues.

One of the most reasonable HPM methodologies is to exploit statistical analysis considering process parameters. It is an extension of SSTA, through which we can generate the mapping function between the measurements and the target timing via PVs. For example, Liu and Sapatnekar [45] derived the estimation of the amounts of spatial variations on a chip by allocating the same type of ring oscillators and measuring them. Later, they proposed the heuristic algorithm of synthesizing replica of a timing critical path that maximizes correlation with a target timing [46]. Chan, Gupta, Kahng, and Lai [47], on the other hand, observed that sensitivity to PVs of timing critical paths forms clusters and suggested to insert one representative ring oscillator per each of



(a)



(b)

Figure 1.4: (a) Hierarchy of process variations. Names in the red, orange, and yellow boxes represent variations of physical parameters, electrical parameters, and circuit timings, respectively. (b) Taxonomy of process variations. HPM methodologies exploit the property of die-to-die (or global random) variations.

them. The biggest merit of these approaches is the straightforwardness and simplicity in interpreting the relation model, which enables further analysis and optimization. However, they assumed that a target timing changes linearly to PVs in common, which is no longer valid in an NTV regime due to large and nonlinear performance variations, as shown in Figure 1.5.

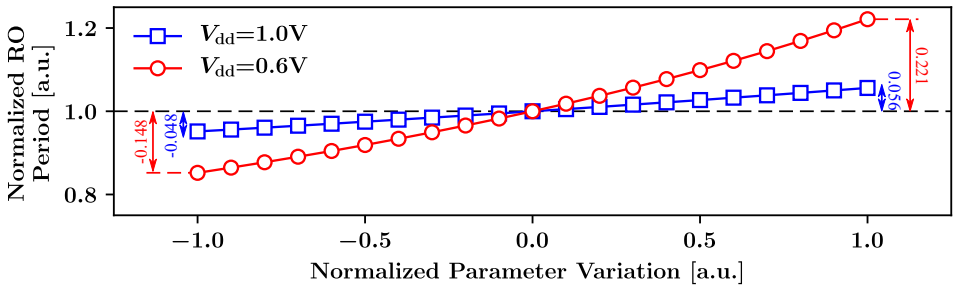


Figure 1.5: Change of an inverter-based ring oscillator period with the variation of one specific process parameter for V_{dd} of 1.0V (super-V_{th} regime) and 0.6V (NTV regime) obtained from SPICE simulation and a 28nm industry PDK. Difference between the periods at both extremes increases to 3.38× when V_{dd} decreases from 1.0V to 0.6V. Note that the normalized variations -1.0 and +1.0 indicate -3σ and $+3\sigma$ value of the parameter variation, respectively.

HPM methodologies combined with advanced machine learning techniques have emerged recently. The most distinguishing feature is that it only requires a dataset consisting of a sufficient number of measurement data and corresponding timings. Mu, Chao, Chen, and Wang [48] selected important parameters among measured ring oscillator frequencies and their polynomial terms and generated a dataset by paring them with the corresponding f_{max} together. After that, they trained an f_{max} prediction model using the dataset through Bayesian linear regression. Sadi, Kannan, Winemberg, and Tehranipoor [49], on the other hand, exploited timing slack sensors as monitoring circuits and compared the results of applying various kinds of machine learning techniques for speed binning of a chip. Though these approaches do not require any

information about process parameters, it is hard to interpret trained models like other machine learning techniques. Hence further analysis and optimization for improving prediction quality could be a daunting task. For example, engineers should retrain the model every time for exploring the impact of the number of monitoring circuit instances on prediction pessimism. In addition, Sadi *et al.* did not provide enough evidence to support the efficacy of their slack sensor insertion algorithm theoretically and empirically.

1.3 Asynchronous Pipeline Controller

Synchronous circuit design style has contributed to the proliferation of VLSI systems and the rapid advancement of the semiconductor industry over the past few decades. One of the main factors of success is the simplicity in the synchronization of all components in systems through a global clock network. However, with the advancement of the semiconductor manufacturing process and the rise of the demand for faster and more energy-efficient electronic circuits in emerging domains, e.g., deep neural networks [26], various issues on a global clock network become more and more critical for VLSI system designers. For example, the technique of fine-tuning clock skew has been widely used for implementing high-performance circuits [50, 47], but its application has become much hard as the noise and delay variability caused by process–voltage–temperature (PVT) variation increases [51]. In addition, reducing the power consumption on a clock network is faced with its limitation due to the stagnation of scaling [52, 53]. In this respect, many researchers are pessimistic about the use of the conventional synchronous circuit design style in the future [54].

An asynchronous circuit is one of the attractive alternatives to the synchronous circuit design style since it is able to cope with the limitations. Contrary to the synchronization mechanism used in a global clock network in synchronous circuits, asynchronous circuits exploit handshaking protocol for the communication between circuit

components, by which they consume less dynamic power and operate at a higher frequency than their synchronous counterparts in general [55, 56]. For a synchronous circuit, a global clock network consumes a large proportion of the total dynamic power, and its performance, i.e., the minimum clock period, of the whole system relies on the most timing critical path. On the other hand, activation of an asynchronous circuit is independent of clock time, and thus the power-hungry clock network is never needed.

Based on this property, from the 1950s to 1970s, several leading asynchronous high-performance processors were implemented by some universities, including the University of Illinois at Urbana-Champaign (ILLIAC in 1952 and ILLIAC II in 1962) and University of Manchester (Atlas in 1962 and MU-5 in 1974). From the 1980s, modern single-chip or SoC style asynchronous microprocessors were mainly designed by Martin's group at Caltech (CAM in 1988 [57] and MiniMIPS in 1997 [58]), Furber's group at the University of Manchester (Amulet1 in 1993 [59], Amulet2e in 1997 [60], and Amulet3i in 2000 [61]), and Nanya's group at Tokyo Institute of Technology (TITAC-1 in 1994 [62] and TITAC-2 in 1997 [63]). Meanwhile, for the commercial purpose, the asynchronous 8051 microcontroller from Philips [64] was sold over 700 million copies with the use of the first fully-automated industrial-strength asynchronous design flow (Tangram, later Haste) [65]. Recently, with the increase of interest in deep learning, neuromorphic processors including IBM's TrueNorth [66], Stanford University's Neurogrid [67], and the University of Manchester's SpiNNaker [68] used asynchronous interconnection networks to integrate massively-parallel processing elements efficiently. However, one of the barriers to the adoption of an asynchronous handshaking mechanism is a large area of a handshaking controller. One notable method to lower the barrier is employing handshaking controller templates [69, 70, 71], and those adapted to pipeline structure are widely used for high-performance applications [72] in particular.

Pipeline controller templates can be sorted into two categories depending on the use of dynamic logics in order to enhance circuit performance. For example, Suther-

land and Fairbanks [73] customized their dynamic logics and inserted them for fast data transmission, and Singh and Nowick [74] proposed the pipeline controller template with full capacity storage using dynamic logics. Furthermore, Fant and Brandt [75], Martin and Nyström [76], and Xia *et al.* [77] suggested the methods of employing dynamic logics with dual outputs. However, their solutions require substantial care or experience to fit into industrial design flows. Though some researchers, e.g., [78, 79], devised CAD tools for optimizing asynchronous circuits in the past, they assumed a restricted cell library or a specific design style, which is hard to integrate with the conventional one.

On the other hand, implementing asynchronous circuits using purely static logics, e.g., [80, 81, 3, 82], is relatively easy to exploit current well-established EDA toolchain and standard cell libraries. For example, Sutherland [80] employed his custom latch (i.e., capture-pass) and a C-element in his pipeline controller template, and Singh and Nowick [81] devised MOUSETRAP, a high-performance pipeline controller template using a normally-transparent latch. Recently, on top of [81], Ho and Chang [3] achieved a significant reduction on dynamic power consumption in datapath by blocking glitches in data transmission using a normally-opaque latch. Toan, Tung, and Lee [82] slightly ameliorated its energy efficiency and performance using a C-element, but the underlying operation is identical to that in [3].

Meanwhile, the performance gap among manufactured chips has grown a lot over the past years due to operation at low V_{dd} and enlargement of process variations. Figure 1.2(a) shows the drastic increase of the performance gap caused by V_{dd} scaling for 28nm process technology, and the curves in Figure 1.6 indicate that the maximum operating frequency can be increased by more than $10\times$ when the temperature changes from $0\,^{\circ}\text{C}$ to $75\,^{\circ}\text{C}$ for the sub-V_{th} regime. Various kinds of post-silicon tuning techniques have been proposed and used to contract this wide performance gap through adjusting delay tunable components on individual chips based on their physical properties. One representative method is clock skew tuning, which resolves timing

violations by inserting delay circuits like adjustable delay buffers (ADBs) to control local clock skews. For example, Figure 1.7(a) describes a capacitor bank based ADB implementation [8], and ARM used a long delay chain of ring oscillators called tunable delay stages (TDSs) shown in Figure. 1.7(b) for tracking temperature variation in the system [9].

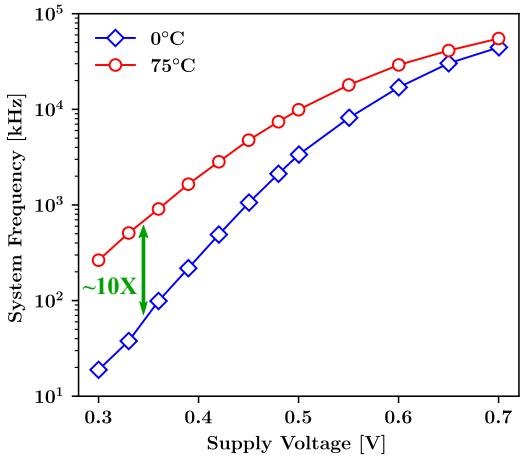


Figure 1.6: Change of system frequency with V_{dd} scaling when the temperatures are $0\,^{\circ}\mathrm{C}$ and $75\,^{\circ}\mathrm{C}$ for 65nm process technology.

For a bundled-data protocol based asynchronous circuit, a receiver should not accept a request signal before arriving data signals from its sender. To ensure this, most of the previous researches focused on the optimization and logical operation of pipeline controller templates with the assumption of using these post-silicon tunable delay circuits [70, 3]. As a result, they required a large number of delay buffers to intentionally provide a long delay path on each pipeline stage, causing a considerable area overhead. Figure 1.8 demonstrates the change of the whole controller and delay buffer area as the required delay for satisfying handshaking communication increases. Except for some kinds of circuits like FIFO buffers, the red curve in Figure 1.8 indicates that the delay buffers occupy a substantial portion of the whole pipeline controller as the required delay increases.

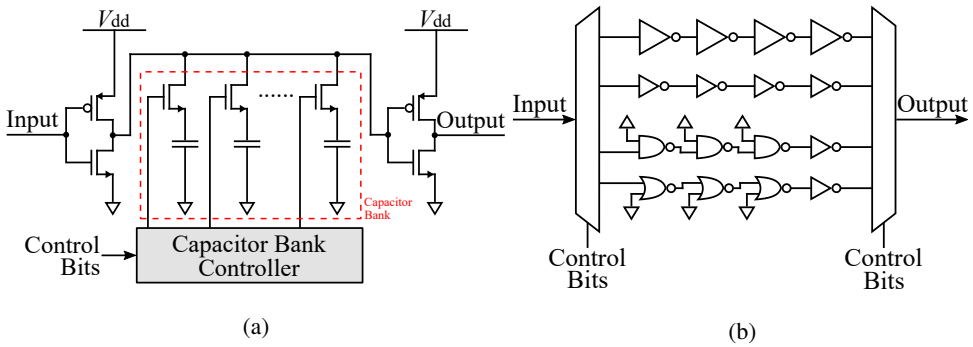


Figure 1.7: (a) The circuit structure of a capacitor bank based adjustable delay buffer (ADB) [8]. The delay depends on the number of capacitors turned on in its capacitor bank. (b) The circuit structure of a tunable delay stage (TDS) used in [9]. The delay can be manipulated by the selection signals that enter the multiplexers and demultiplexers.

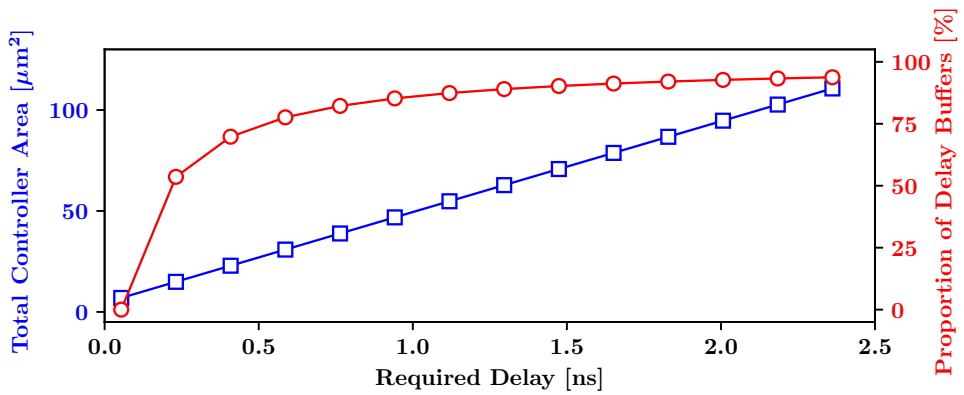


Figure 1.8: Change of the area of an asynchronous pipeline controller when all delay circuits are delay buffer chains (blue curve) and the proportion of delay buffers (red curve) on a pipeline stage. I measured the area and delay according to 45nm NanGate Open Cell library [4].

1.4 Contributions of this Dissertation

Timing analysis is one of the necessary steps in the development of a semiconductor circuit. Furthermore, it is increasingly important as PVT variation increases in advanced process technologies. For example, it is required to introduce an accurate timing analysis technique based on a more realistic timing model for reducing a bit of timing pessimism in the design process (Chapter 2). Even in the post-silicon phase, precise and efficient timing prediction methodology is also essential to apply various kinds of adaptive low-power techniques (Chapter 3). Besides, sophisticated optimization based on timing analysis is one of the top priorities in emerging circuit structures. (Chapter 4). In this dissertation, I address three problems related to timing analysis and optimization in logic and physical synthesis. The contributions of each chapter are as follows.

- In Chapter 2, I propose the analysis method under flexible flip-flop timing model derived from a comprehensive interval analysis of a clock-to-Q delay, with a demonstration of applications to two examples. I also formulate applications of my flexible flip-flop timing analysis into convex optimization problems: the problem of finding the minimum clock period of a given circuit and the clock skew scheduling problems for maximizing the worst and total timing slacks. Lastly, I suggest a technique for improving the speed and scalability of solving them.

- In Chapter 3, I propose a holistic HPM methodology, from design to post-silicon phase, for handling wide and nonlinear performance variation in NTC and advanced process. I first reduce the problem of finding an efficient configuration of monitoring circuits into the optimal experiment design problem. I also propose the target timing prediction flow by combining the statistical estimation of FEOL and BEOL PVs and a neural network based inference model with uncertainty learning. Lastly, I suggest the calibration flow throughout typical IC design flow

via transfer learning and employ surrogate models for avoiding time-consuming SPICE simulations.

- In Chapter 4, I propose a technique of synthesizing an asynchronous pipeline controller in a way to share delay buffers among setup timing paths on pipeline stages (i.e., delay path sharing), so that the total delay buffers should be minimally allocated, formulating the allocation problem into linear programming. By extending the delay path sharing concept (i.e., delay path reusing), I devise a new area-efficient delay circuit structure called delay path unit (DPU) and propose an in-depth synthesis flow of an asynchronous pipeline controller using DPU.

Chapter 2

ANALYSIS AND OPTIMIZATION CONSIDERING FLEXIBLE FLIP-FLOP TIMING MODEL

2.1 Preliminaries

2.1.1 Terminologies

Let me consider the circuit consisting of two flip-flops FF_i and FF_j shown in Figure 2.1, and let $x(i)$, $x(j)$, $D_{i,j}^{max}$, and $D_{i,j}^{min}$ in the figure be clock arrival times at FF_i and FF_j and maximum and minimum propagation delays of the combinational logic from FF_i to FF_j, respectively. Then the static timing analysis terms are formally defined as follows:

- *Setup skew* $t_{skew}^{su}(j)$: The setup skew $t_{skew}^{su}(j)$ of FF_j is the time interval ending at the clock's active edge of the flip-flop during which the input data to FF_j is stable.

- *Hold skew* $t_{skew}^{h}(j)$: The hold skew $t_{skew}^{h}(j)$ of FF_j is the time interval starting at the clock's active edge of the flip-flop during which the input data to FF_j is stable.

- *Clock-to-Q delay* $t_{c2q}(j)$: The clock-to-Q delay $t_{c2q}(j)$ of FF_j refers to the time lapse between the clock's arrival edge of the flip-flop and the time to register the input data to the output of FF_j. If FF_j's setup skew $t^{su}_{skew}(j)$, hold skew $t^{h}_{skew}(j)$, and clock-to-Q delay surface $\mathcal{F}_j$ are given, $t_{c2q}(j)$ could be expressed as

$$t_{c2q}(j) = \mathcal{F}_j\big(t^{su}_{skew}(j), t^{h}_{skew}(j)\big).$$

- *Setup time* $t^{su}_{time}(j)$: The setup time value $t^{su}_{time}(j)$ of FF_j is a lower bound of the setup skew of FF_j set by a designer to make sure a successful latching at FF_j.

- *Hold time* $t^{h}_{time}(j)$: The hold time value $t^{h}_{time}(j)$ of FF_j is a lower bound of the hold skew of FF_j set by a designer to make sure a successful latching at FF_j.

- *Setup slack* $t^{su}_{slk}(j)$: The setup slack $t^{su}_{slk}(j)$ of FF_j is the extra setup skew available beyond the setup time $t^{su}_{time}(j)$ of FF_j with respect to the input data, i.e.,

$$t^{su}_{slk}(j) = t^{su}_{skew}(j) - t^{su}_{time}(j).$$

- *Hold slack* $t^{h}_{slk}(j)$: The hold slack $t^{h}_{slk}(j)$ of FF_j is the extra hold skew available beyond the hold time $t^{h}_{time}(j)$ of FF_j with respect to the input data, i.e.,

$$t^{h}_{slk}(j) = t^{h}_{skew}(j) - t^{h}_{time}(j).$$

- *Worst slack* t^{w}_{slk}: The worst slack t^{w}_{slk} of a circuit is the minimum value of the setup and hold slacks for all flip-flops in the circuit, i.e.,

$$t^{w}_{slk} = \min_{FF_j} \big\{t^{su}_{slk}(j), t^{h}_{slk}(j)\big\}.$$

- *Total slack* t^{tot}_{slk}: The total slack t^{tot}_{slk} of a circuit is the summation of the minimum slack values all flip-flops have, i.e.,

$$t^{tot}_{slk} = \sum_{FF_j} \min \big\{t^{su}_{slk}(j), t^{h}_{slk}(j)\big\}.$$

Figure 2.1 shows pictorial representation of the terminologies with timing diagram.

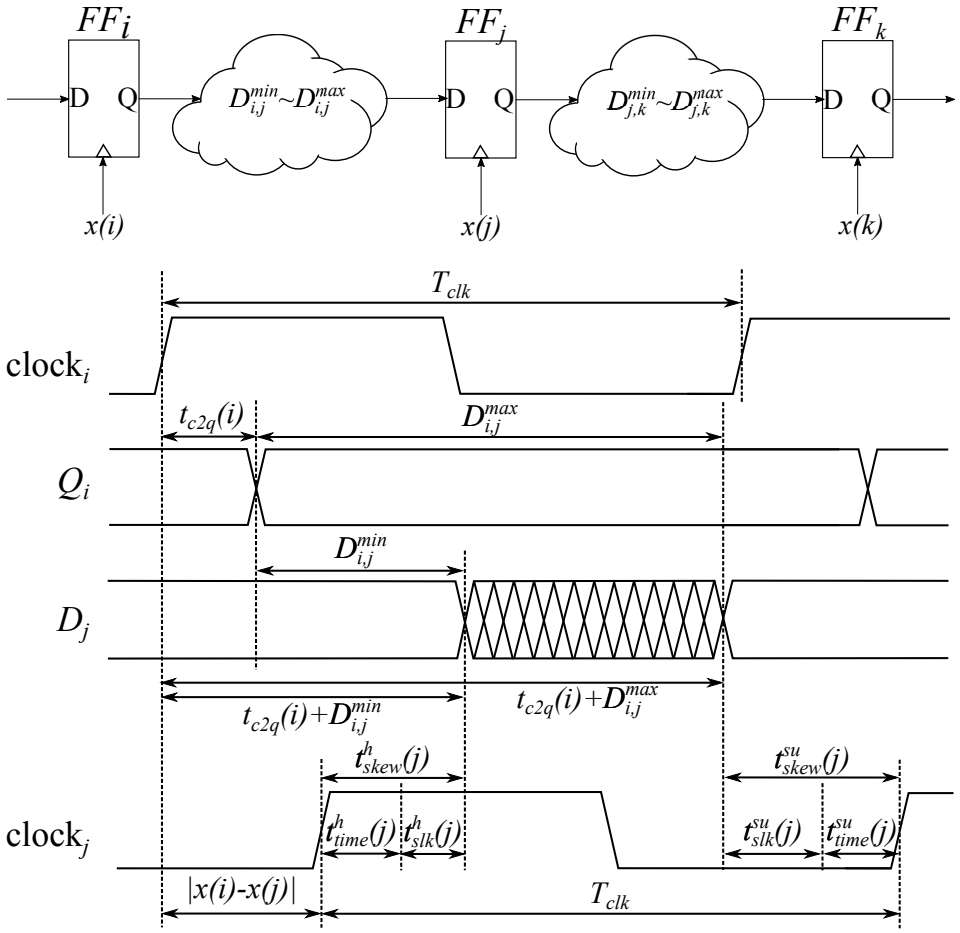


Figure 2.1: A simple circuit and its timing diagram.

2.1.2 Timing Analysis

A target circuit is first decomposed into the set of all timing paths and the maximum and the minimum signal propagation delays of each path are calculated. Then, timing validation at all flip-flops and primary inputs and outputs is checked. For simplicity, consider two fundamental timing paths: a data path and a clock path. A data path is a path that propagates logical signals. It starts from a primary input port or an output data pin of a flip-flop and ends at an input data pin of a flip-flop or a primary output port. A clock path, on the other hand, is a path that propagates clock signals. It starts from a clock source port and ends at each flip-flop's input clock pin. The delay of a clock path includes not only the delays of buffers or inverters but also interconnect delays.

In a synchronous digital system, there are two timing errors related to flip-flops or latches: a setup time violation and a hold time violation. A setup time violation occurs when the data signal arrives *too late* relative to the active clock transition, while a hold time violation occurs when it arrives *too soon*. To avoid the timing violations, designers introduce *a setup time constraint* $t^{su}_{skew}(j) \geq t^{su}_{time}(j)$ (or equivalently, $t^{su}_{slk}(j) \geq 0$) for each FF_j, and *a hold time constraint* $t^{h}_{skew}(j) \geq t^{h}_{time}(j)$ (or equivalently, $t^{h}_{slk}(j) \geq 0$) for each FF_j. For example, for the circuit in Figure 2.1, the setup and hold skews of FF_j are computed by

$$t^{su}_{skew}(j) = \big(T_{clk} + x(j)\big) - \big(x(i) + t_{c2q}(i) + D_{i,j}\big),$$
$$t^{h}_{skew}(j) = \big(x(i) + t_{c2q}(i) + D_{i,j}\big) - x(j),$$

where $D^{min}_{i,j} \leq D_{i,j} \leq D^{max}_{i,j}$. The computed values are compared with the preset values of the setup time and the hold time of FF_j. If both of the skews are larger, the analyzer will ensure a correct operation at FF_j.

2.1.3 Clock-to-Q Delay Surface Modeling

One of the most important and basic tasks for flexible flip-flop timing based analysis and optimization is characterizing a clock-to-Q delay surface. A clock-to-Q delay surface is a three dimensional graph that describes the change of the clock-to-Q delay of a flip-flop for various pairs of setup and hold skew values. Its x, y, and z axes denote a flip-flop's setup skew, hold skew, and the corresponding clock-to-Q delay, respectively, as shown in Figure 2.2.

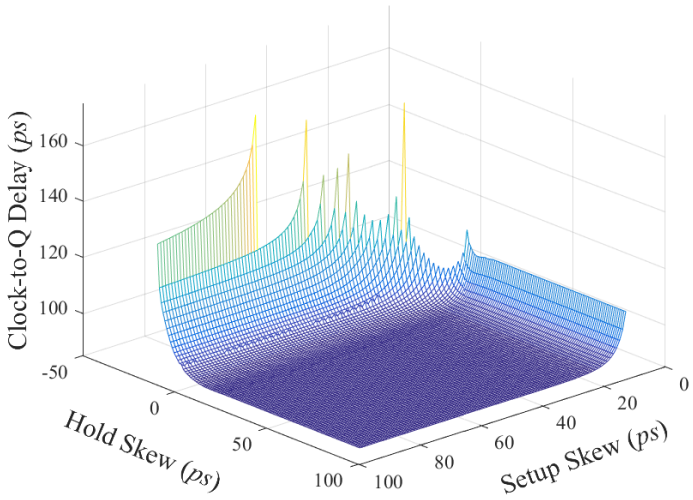


Figure 2.2: View of a clock-to-Q delay surface. All data of $(t^{su}_{skew}, t^{h}_{skew}, t_{c2q})$ are extracted at every 1ps of timing points through SPICE simulations with 45nm NanGate Open Cell Library [4], where t^{su}_{skew} and t^{h}_{skew} are characterized over the range of 0ps$\sim$100ps and -20ps$\sim$100ps, respectively.

For modeling a clock-to-Q delay surface, Chen, Li, and Schlichtmann [20] tried to fit the SPICE simulation results into the following analytic function

$$t_{c2q} = a_0 + \frac{a_1}{t^{su}_{skew} - s_0} + \frac{a_2}{t^{h}_{skew} - h_0} \tag{2.1}$$

in which a_0, a_1, a_2, s_0, and h_0 are the fitting coefficients. It is a nonlinear fitting problem as the form of the function suggests, and it can be effective to use nonlinear least

squares, which is used to fit a set of observations with a model consisting of unknown parameters. Contrary to [20], Kahng and Lee [21] suggested to exploit linearly approximated relations between setup skews and clock-to-Q delays for given hold skews, or hold skews and clock-to-Q delays for given setup skews. Conventionally this type of task is categorized as a linear fitting problem, and various objectives can be utilized for the problem. One possible direction is linear least squares, which minimizes the sum of all residuals for given data points.

More accurate clock-to-Q delay surface modeling than that in Equation (2.1) would also be possible. For instance, the latest convex regression techniques such as [83, 84, 85, 86], and other various works of applied mathematics communities can be exploited for clock-to-Q delay surface modeling. It should be noted that my timing analyzer can use any existing clock-to-Q delay surface model as long as it is convex.

2.2 Clock-to-Q Delay Interval Analysis

In this section, I first derive my clock-to-Q delay interval analysis in Section 2.2.1. After that, I apply it to two representative applications of flexible flip-flop timing model: (1) finding minimum clock period of a given circuit design, (2) clock skew scheduling for improving worst slack, clock period, and total slack. Note that (1) is an analysis problem while (2) is an optimization problem. In Section 2.2.2, I add two additional constraints, i.e., an interacting constraint and a clock-to-Q delay constraint, and then formulate the analysis problem into a convex programming based on the constraints, in Section 2.2.3. In addition, I apply my concept to the clock skew scheduling problem in Section 2.2.4. Finally, I explain my speedup technique for those applications in Section 2.2.5.

2.2.1 Derivation

Since the clock-to-Q delay at a certain clock cycle depends on the setup skew at the previous clock cycle and the hold skew at the current clock cycle, the clock-to-Q delay at cycle c of FF_j in Figure 2.1 can be written as

$$\begin{aligned}
t_{skew}^{su,(c-1)}(j) &= \big(T_{clk} + x(j)\big) - \big(x(i) + t_{c2q}^{(c-1)}(i) + D_{i,j}^{max,(c-1)}\big),\\
t_{skew}^{h,(c)}(j) &= \big(x(i) + t_{c2q}^{(c)}(i) + D_{i,j}^{min,(c)}\big) - x(j),\\
t_{c2q}^{(c)}(j) &= \mathcal{F}_j\big(t_{skew}^{su,(c-1)}(j), t_{skew}^{h,(c)}(j)\big).
\end{aligned}$$

Let me assume that all logical operations at every clock cycle are independent to each other as being done in conventional static timing analysis. Then it is possible to have the case that $t_{c2q}^{(c)}$ is very large while $t_{c2q}^{(c+1)}$ is very small or vice versa; therefore, the concept of the convergence in the pointwise manner in [20, 21, 22, 2] would not result an accurate solution, and it is more natural to consider the convergence based on clock-to-Q delay intervals.

Let $t_{c2q}^{min}(i)$ and $t_{c2q}^{max}(i)$ be the minimum and maximum clock-to-Q delays of FF_i, respectively. Then, the minimum setup skew $t_{skew}^{su,min}(j)$ and the minimum hold skew $t_{skew}^{h,min}(j)$ at FF_j are

$$\begin{aligned}
t_{skew}^{su,min}(j) &= \big(T_{clk} + x(j)\big) - \big(x(i) + t_{c2q}^{max}(i) + D_{i,j}^{max}\big),\\
t_{skew}^{h,min}(j) &= \big(x(i) + t_{c2q}^{min}(i) + D_{i,j}^{min}\big) - x(j).
\end{aligned} \tag{2.2}$$

The maximum setup skew $t_{skew}^{su,max}(j)$ and the maximum hold skew $t_{skew}^{h,max}(j)$ at FF_j are also expressed similarly. By reasoning the more delay is required to charge or discharge the internal capacitor of a flip-flop as the setup skew and the hold skew diminish, the minimum and the maximum clock-to-Q delays at FF_j can be expressed as

$$\begin{aligned}
t_{c2q}^{min}(j) &= \mathcal{F}_j\big(t_{skew}^{su,max}(j), t_{skew}^{h,max}(j)\big),\\
t_{c2q}^{max}(j) &= \mathcal{F}_j\big(t_{skew}^{su,min}(j), t_{skew}^{h,min}(j)\big),
\end{aligned} \tag{2.3}$$

where $\mathcal{F}_j$ is the clock-to-Q delay surface of FF_j. By the same token, the procedure for deriving the interval of the clock-to-Q delay at FF_k of the circuit in Figure 2.1 can

be expressed as

$$
\begin{aligned}
t_{skew}^{su,min}(k) &= \big(T_{clk} + x(k)\big) - \big(x(j) + t_{c2q}^{max}(j) + D_{j,k}^{max}\big),\\
t_{skew}^{h,min}(k) &= \big(x(j) + t_{c2q}^{min}(j) + D_{j,k}^{min}\big) - x(k),\\
t_{c2q}^{min}(k) &= \mathcal{F}_k\big(t_{skew}^{su,max}(k), t_{skew}^{h,max}(k)\big),\\
t_{c2q}^{max}(k) &= \mathcal{F}_k\big(t_{skew}^{su,min}(k), t_{skew}^{h,min}(k)\big).
\end{aligned}
\tag{2.4}
$$

By repeating this process for all flip-flops, the intervals of their clock-to-Q delays as well as the timing relations between their parameters being involved can be obtained easily.

Now let me find out the impacts of the analysis methods, i.e., the conventional pointwise convergence based analysis in [20, 21, 22, 2] and my clock-to-Q delay interval based analysis, with two example circuits. For clarity, I will denote clock-to-Q delays obtained from pointwise convergence based analysis with an asterisk.

Case 1: The circuit of Figure 2.1. Let me suppose that FF_i has a specific value of a clock-to-Q delay, i.e., $t_{c2q}(i)$, initially. From the clock-to-Q delay of FF_i, previous works compute the setup skew and the hold skew of FF_j, from which the clock-to-Q delay of FF_j $\big(t_{c2q}^{*}(j)\big)$ is determined and marked with a red dot in Figure 2.3(a). Likewise, the worst skew scenario of a timing path to FF_k together with $t_{c2q}^{*}(j)$ will provide the clock-to-Q delay of FF_k $\big(t_{c2q}^{*}(k)\big)$, as illustrated in Figure 2.3(b). However, FF_j can have a smaller clock-to-Q delay than $t_{c2q}^{*}(j)$ in reality because of the independence between the two clock-to-Q delays used in the calculation of the setup skew and the hold skew of FF_j; in other words, all points in the blue region of Figure 2.3(a) can be a pair of the setup skew and the hold skew of FF_j. The minimum setup skew $\big(t_{skew}^{su,min}(k)\big)$ and the minimum hold skew $\big(t_{skew}^{h,min}(k)\big)$ at FF_k are then calculated like Equation (2.4), and I can easily deduce that the gap between $t_{c2q}^{*}(j)$ and $t_{c2q}^{min}(j)$ causes reduction of the minimum hold skew of FF_k in Figure 2.3(b). (Note that the clock-to-Q delay value increases as the point gets closer to the separation curve.) As a result, $t_{c2q}^{max}(k)$ would be located below the curve, and therefore, against

the expectation based on the previous thought, the circuit may run into the danger of falling to the timing failure region.

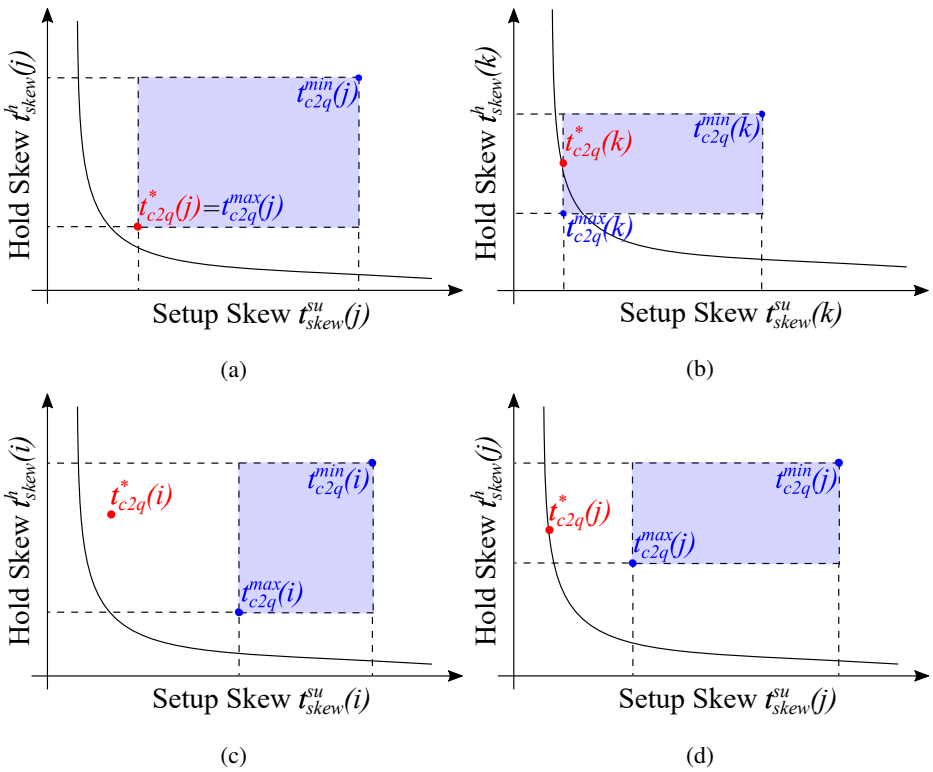


Figure 2.3: (a)-(b) Timing analyses of FF_j and FF_k of the circuit in Figure 2.1. (c)-(d) Timing analyses of FF_i and FF_j of the circuit in Figure 2.4. In all figures, the outcomes based on the previous notion and mine are in red and blue, respectively. Each plot represents the top view of the clock-to-Q delay surface of the corresponding flip-flop, and each curve represents the boundary between success and failure regions of latching. Therefore, the points located above the curve are safe while the others are in danger.

Case 2: The circuit of Figure 2.4. Let us suppose that the circuit in Figure 2.4 has a simple feedback loop. Unlike the previous case, through the iteration, the location of the clock-to-Q delay of FF_i $\left(t^*_{c2q}(i)\right)$ may be far from the blue box bounded by the

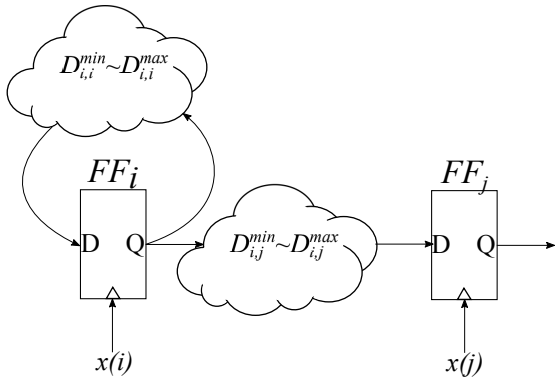


Figure 2.4: A simple circuit.

minimum and the maximum of the setup skews and the hold skews of FF_i, as shown in Figure 2.3(c). If $t^*_{c2q}(i) \geq t^{max}_{c2q}(i)$, the minimum setup skew of FF_j calculated from $t^{max}_{c2q}(i)$ will be larger than that from $t^*_{c2q}(i)$ as shown in Figure 2.3(d) according to Equation (2.2). Similarly, the minimum hold skew of FF_j obtained from $t^{min}_{c2q}(i)$ is smaller than that from $t^*_{c2q}(i)$, due to $t^*_{c2q}(i) \geq t^{min}_{c2q}(i)$ if $t^*_{c2q}(i) \geq t^{max}_{c2q}(i)$, as shown in the same figure. As a result, contrary to the case of the circuit in Figure 2.1, there can be a room for further timing optimization, e.g. increasing the clock frequency.

2.2.2 Additional Constraints

(1) ***Formulating interacting constraints of setup and hold times***: Setup and hold times are treated independently under fixed flip-flop timing model. However, in flexible flip-flop timing model based analysis, I should consider the interaction of them through clock-to-Q delay surface $\mathcal{F}\left(t^{su}_{skew}, t^{h}_{skew}\right)$. A flip-flop is able to latch its data with a certain pair of setup and hold skews, so there exists a boundary curve among the pairs. To represent this curve and include it in my formulation, I create the function $\mathcal{G}\left(t^{su}_{skew}, t^{h}_{skew}\right)$ and let $\mathcal{G}$ has negative values in the feasible region while positive values on the opposite side. Then the setup and hold times I can feasibly set with no

timing violations are in the region including the curve that satisfies

$$\mathcal{G}\left(t^{su}_{skew}, t^{h}_{skew}\right) \leq 0,$$

as shown in Figure 2.5. Note that since there is abundant choice of selecting setup and hold times, i.e., all points satisfying the inequality, the exploration of alternatives will enable the timing analysis and optimization to be more flexible. One simple method to create the function $\mathcal{G}$ is to set

$$\mathcal{G}\left(t^{su}_{skew}, t^{h}_{skew}\right) = \mathcal{F}\left(t^{su}_{skew}, t^{h}_{skew}\right) - t^{upr}_{c2q}$$

with an upper bound of a clock-to-Q delay value t^{upr}_{c2q}.

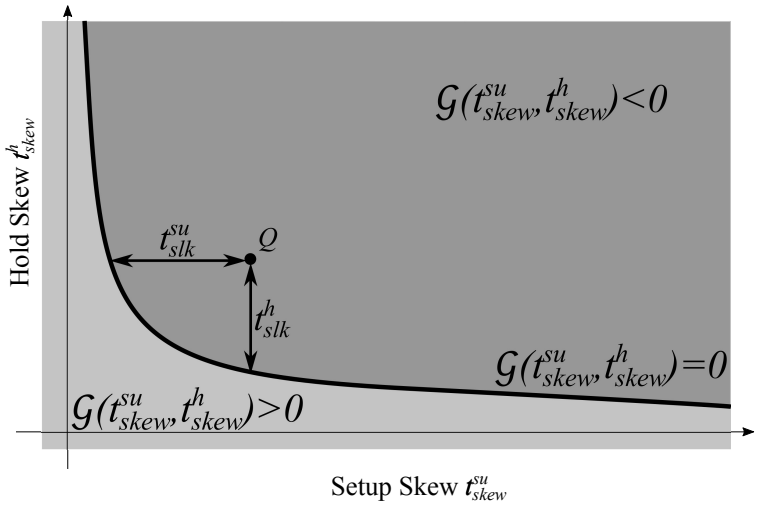


Figure 2.5: Description of the boundary curve $\mathcal{G}\left(t^{su}_{skew}, t^{h}_{skew}\right) = 0$. The upper right part, represented by $\mathcal{G}\left(t^{su}_{skew}, t^{h}_{skew}\right) < 0$, is safe for setting setup and hold times, while setting them in the lower left region, i.e., $\mathcal{G}\left(t^{su}_{skew}, t^{h}_{skew}\right) > 0$, is infeasible due to the failure of a latching. If the minimum setup and hold skews of a flip-flop are located at point Q, the setup and hold slacks of that flip-flop are the distances from Q to the curve $\mathcal{G}\left(t^{su}_{skew}, t^{h}_{skew}\right) = 0$ along x and y axes, respectively.

(2) ***Formulating clock-to-Q delay constraints***: The clock-to-Q delay constraint at FF_j

is exactly the boundary constrained by Equation (2.3), which can be rewritten as:

$$\mathcal{F}_j\big(t_{skew}^{su,min}(j), t_{skew}^{h,min}(j)\big) \leq t_{c2q}^{max}(j),$$
$$t_{c2q}^{min}(j) \leq \mathcal{F}_j\big(t_{skew}^{su,max}(j), t_{skew}^{h,max}(j)\big).$$

2.2.3 Analysis: Finding Minimum Clock Period

The problem in this subsection is to find a minimum clock period at which a given circuit can operate with no timing violations for given clock arrival times. To consider the combination of setup and hold time constraints, let me assume that N flip-flops $FF_{(i,1)}, \cdots, FF_{(i,N)}$ have combinational logic paths to FF_j, as shown in Figure 2.6. Each path from $FF_{(i,n)}$ to FF_j ($n = 1, \cdots, N$) has the minimum setup and hold skews

$$t_{skew}^{su,min}(i,n,j) = \Big(T_{clk} + x(j)\Big) - \Big(x(i,n) + t_{c2q}^{max}(i,n) + D_{(i,n,j)}^{max}\Big),$$
$$t_{skew}^{h,min}(i,n,j) = \Big(x(i,n) + t_{c2q}^{min}(i,n) + D_{(i,n,j)}^{min}\Big) - x(j),$$

where $t_{skew}^{su,min}(i,n,j)$ and $t_{skew}^{h,min}(i,n,j)$ are the minimum setup and hold skews of the path from $FF_{(i,n)}$ to FF_j, respectively. The maximum setup and hold skews of each path can also be calculated similarly. Since all paths are independent to each other in static timing analysis, the minimum setup and hold skews of FF_j, i.e., $t_{skew}^{su,min}(j)$ and $t_{skew}^{h,min}(j)$, should be less than all the minimum setup and hold skews, and the maximum setup and hold skews of FF_j, i.e., $t_{skew}^{su,max}(j)$ and $t_{skew}^{h,max}(j)$, should also be greater than all the maximum setup and hold skews; therefore it can be expressed as

$$t_{skew}^{su,min}(j) \leq \min_{n=1,\cdots,N}\big\{t_{skew}^{su,min}(i,n,j)\big\},$$
$$t_{skew}^{h,min}(j) \leq \min_{n=1,\cdots,N}\big\{t_{skew}^{h,min}(i,n,j)\big\},$$
$$t_{skew}^{su,max}(j) \geq \max_{n=1,\cdots,N}\big\{t_{skew}^{su,max}(i,n,j)\big\},$$
$$t_{skew}^{h,max}(j) \geq \max_{n=1,\cdots,N}\big\{t_{skew}^{h,max}(i,n,j)\big\}.$$

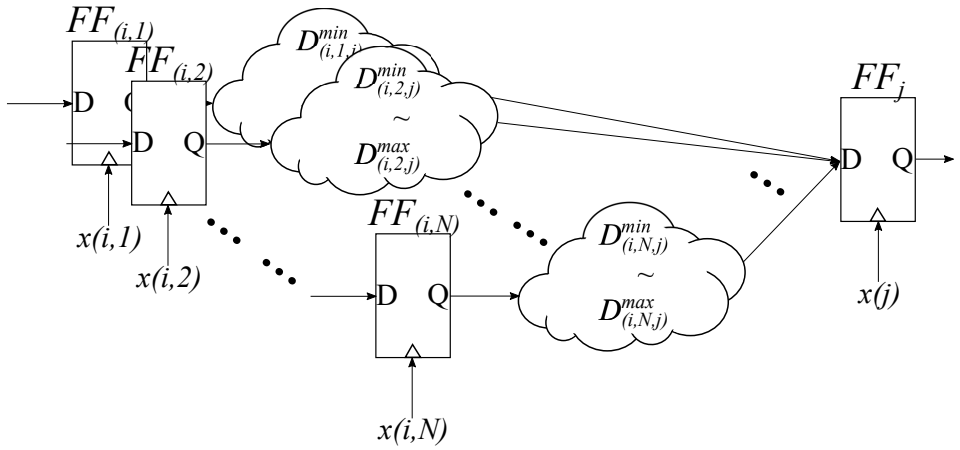


Figure 2.6: A circuit illustrating my derivation of formulation for the problem in Section 2.2.3. For clarity, only three paths are shown in the figure.

In addition, the following timing analysis constraints, i.e., the interacting constraints of setup and hold times and the clock-to-Q delay constraints, should satisfy

$$\mathcal{G}_j\big(t^{su}_{skew}(j), t^{h}_{skew}(j)\big) \leq 0,$$
$$t^{min}_{c2q}(j) \leq \mathcal{F}_j\big(t^{su}_{skew}(j), t^{h}_{skew}(j)\big) \leq t^{max}_{c2q}(j)$$

for each flip-flop FF_j.

At the same time, it is necessary to check if the solution space constrained by the inequalities with functions $\mathcal{G}$ and $\mathcal{F}$ is convex for reliable and efficient exploration. The following three assumptions support the convexity:

- *Assumption 1*. $\mathcal{G}$ for every flip-flop is a convex and monotonic non-increasing.
- *Assumption 2*. $\mathcal{F}$ for every flip-flop is a convex and monotonic non-increasing.
- *Assumption 3*. The lower bound of clock-to-Q delay for every flip-flop equals to the minimum possible clock-to-Q delay of the flip-flop.

Assumption 1 is acceptable since $\mathcal{G}$ is defined by a designer. *Assumption 2* is also strongly supported in the light of the shape of a clock-to-Q delay surface, i.e., Figure 2.2, and by the qualitative analysis of the internal operation of flip-flops. Finally,

Assumption 3 does also make sense in that clock-to-Q delay decreases very rapidly as setup and hold skews increase, ultimately approaching the minimum clock-to-Q delay, as shown in Figure 2.2.

As a result, the problem can be transformed into a form of convex programming expressed as

$$
\begin{aligned}
\text{minimize} \quad & T_{clk} \\
\text{subject to} \quad & t_{skew}^{su,min}(j) \leq \big(T_{clk} + x(j)\big) - \big(x(i) + t_{c2q}^{max}(i) + D_{i,j}^{max}\big), \\
& \qquad\qquad\qquad\qquad \forall \text{flip-flop pairs}(i,j), \qquad (2.5) \\
& \mathcal{F}_j\big(t_{skew}^{su,min}(j), t_{skew}^{h,min}(j)\big) \leq t_{c2q}^{max}(j), \quad \forall FF_j, \\
& \mathcal{G}_j\big(t_{skew}^{su,min}(j), t_{skew}^{h,min}(j)\big) \leq 0, \qquad\quad \forall FF_j.
\end{aligned}
$$

T_{clk}, $t_{skew}^{su,min}$, and t_{c2q}^{max} are optimization variables while the others are constants obtained from analyzing the target circuit and characterizing flip-flops in the optimization problem of Equation (2.5). Note that $t_{skew}^{h,min}$ can be computed using clock arrival times already known and *Assumption 3* before solving Equation (2.5); therefore, I can exclude all optimization variables related to $t_{skew}^{h,min}$ and t_{c2q}^{min}.

2.2.4 Optimization: Clock Skew Scheduling

Clock skew scheduling is another representative timing optimization problem which exploits the adjustment of clock arrival times of flip-flops in order to relax the worst and total slacks or increase the clock frequency of a circuit. Relaxation of setup and hold slacks by exploiting clock skew scheduling, in particular, is important for a number of reasons [22, 2]. One reason is that it could resolve setup and hold time violations with little effort to remove them using late-stage ECO knobs, e.g., trial-and-error of gate sizing, threshold voltage swapping. Another reason is that it enables a circuit to be highly tolerant to PVT variation which are frequently occurred in recent nano-scale high speed designs.

In spite of its importance, there are few previous works handled the integration

of clock skew scheduling and flexible flip-flop timing model, and among them, to the best of my knowledge, no work has considered the intervals of clock-to-Q delays of flip-flops yet. I formulate the problem in a form of convex programming with the three assumptions presented in Section 2.2.3. Basic constraints are identical to those in Equation (2.5), but I require additional constraints for expressing slacks and adjustment ranges of clock arrival times.

(1) ***Formulating worst and total slack constraints***: Worst slack is the minimum value among all setup and hold slacks, thus it can simply be formulated as

$$\begin{aligned} t^{w}_{slk} &\leq t^{su}_{slk}(j) = t^{su,min}_{skew}(j) - t^{su}_{time}(j) \quad \forall FF_j, \\ t^{w}_{slk} &\leq t^{h}_{slk}(j) = t^{h,min}_{skew}(j) - t^{h}_{time}(j) \quad \forall FF_j. \end{aligned} \tag{2.6}$$

Equation (2.6) includes $t^{su}_{time}(j)$ and $t^{h}_{time}(j)$, which can be easily calculated from $t^{su}_{skew}(j)$ and $t^{h}_{skew}(j)$, respectively. For an example of the nonlinear analytical model in [20], $\mathcal{G}_j\big(t^{su}_{skew}(j), t^{h}_{skew}(j)\big) = a_0 + a_1/\big(t^{su}_{skew} - s_0\big) + a_2/\big(t^{h}_{skew} - h_0\big) - t^{upr}_{c2q}(j)$ if $\mathcal{G}_j$ is assumed to be a constant clock-to-Q delay curve; thus $t^{su}_{time}(j)$ and $t^{h}_{time}(j)$ corresponding to $t^{su,min}_{skew}(j)$ and $t^{h,min}_{skew}(j)$ are obtained by

$$\begin{aligned} t^{su}_{time}(j) &= \frac{a_1}{t^{upr}_{c2q}(j) - a_0 - a_2/\big(t^{h,min}_{skew}(j) - h_0\big)} + s_0, \\ t^{h}_{time}(j) &= \frac{a_2}{t^{upr}_{c2q}(j) - a_0 - a_1/\big(t^{su,min}_{skew}(j) - s_0\big)} + h_0. \end{aligned} \tag{2.7}$$

Total slack is the summation of minimum slack values all flip-flops have, and the constraint is formulated as

$$t^{tot}_{slk} \leq \sum_{FF_j} \min\big\{t^{su}_{slk}(j), t^{h}_{slk}(j)\big\}, \tag{2.8}$$

where $t^{su}_{slk}(j)$ and $t^{h}_{slk}(j)$ are the same as those in Equation (2.6).

For formulating convex programming, the solution space constrained by all constraints should be a convex set. In the case of the constraints of Equation (2.6), $t^{su}_{time}(j)$ and $t^{h}_{time}(j)$ are convex functions of $t^{h,min}_{skew}(j)$ and $t^{su,min}_{skew}(j)$, respectively, due to *Assumption 2* and the property that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex if and only if the function

$g : \mathbb{R} \to \mathbb{R}$, $g(t) = f(x + tv)$, $\mathbf{dom}\ g = \{t | x + tv \in \mathbf{dom}\ f\}$ is convex in t for any $x \in \mathbf{dom}\ f$ and $v \in \mathbb{R}^n$. Hence, the right hand sides of Equation (2.6) are concave, and consequently, the solution space is a convex set. For the case of a total slack constraint, the right hand side of Equation (2.8) is also concave since minimum of concave functions is concave and summation operator preserves its convexity or concavity; thus the solution space is again a convex set. It should be noted that I do not need to care about the clock-to-Q delay surface model for Equation (2.6) as long as the boundary function G_j is convex.

(2) ***Formulating boundaries of clock arrival times***: Clock signal arrival times cannot be increased or decreased unlimitedly because of the limitation of the amount of available resources, e.g., metal wires, buffers, inverters. The boundary of each clock arrival time x thus should be expressed as

$$\gamma_1(j) \leq x(j) \leq \gamma_2(j), \quad \forall FF_j, \tag{2.9}$$

where $\gamma_1(j)$ and $\gamma_2(j)$ are the lower and the upper bounds of the adjustment amount of clock arrival time of FF_j set by a designer.

In summary, clock skew scheduling problem for maximizing worst slack based on flexible flip-flop timing model and my clock-to-Q delay interval analysis is expressed

as

$$
\begin{aligned}
\text{maximize} \quad & t^{w}_{slk} \\
\text{subject to} \quad & t^{su,min}_{skew}(j) \leq \big(T_{clk} + x(j)\big) - \big(x(i) + t^{max}_{c2q}(i) + D^{max}_{i,j}\big), \\
& \qquad\qquad \forall \text{flip-flop pairs}(i,j), \\
& t^{h,min}_{skew}(j) \leq \big(x(i) + t^{min}_{c2q}(i) + D^{min}_{i,j}\big) - x(j), \\
& \qquad\qquad \forall \text{flip-flop pairs}(i,j), \\
& \mathcal{F}_j\big(t^{su,min}_{skew}(j), t^{h,min}_{skew}(j)\big) \leq t^{max}_{c2q}(j), \qquad \forall FF_j, \\
& t^{w}_{slk} \leq t^{su,min}_{skew}(j) - \mathcal{H}^{su}_{j}\big(t^{h,min}_{skew}(j)\big), \qquad \forall FF_j, \\
& t^{w}_{slk} \leq t^{h,min}_{skew}(j) - \mathcal{H}^{h}_{j}\big(t^{su,min}_{skew}(j)\big), \qquad \forall FF_j, \\
& \gamma_1(j) \leq x(j) \leq \gamma_2(j), \qquad \forall FF_j, \\
& t^{w}_{slk} \geq 0,
\end{aligned}
\tag{2.10}
$$

where $\mathcal{H}^{su}_{j}$ and $\mathcal{H}^{h}_{j}$ are the equations of $t^{su}_{time}(j)$ and $t^{h}_{time}(j)$ derived from Equation (2.7), and optimization variables in Equation (2.10) are t^{w}_{slk}, $t^{su,min}_{skew}$, $t^{h,min}_{skew}$, t^{max}_{c2q}, and x. Note that I can omit interacting constraints of setup and hold times in this formulation since the forth and the fifth constraints can replace them perfectly with the last constraint. All t^{min}_{c2q} are also vanished due to *Assumption 3*. In addition, Equation (2.10) can be exploited for minimizing clock period by substituting T_{clk} for the objective function and including it in the optimization variables; furthermore, total slack maximization is also possible by replacing the objective with the right hand side of Equation (2.8).

2.2.5 Scalable Speedup Technique

To solve Equations (2.5) and (2.10) efficiently, I can exploit various algorithms, e.g., the interior-point polynomial time method [87], which is known as one of the most efficient techniques in theory and practice to solve many convex optimization problems including linear programming, second-order cone programming, semidefinite programming, etc. However, as a circuit size grows and the number of flip-flops explodes,

the interior-point method will suffer from the run time issue as well as the capacity problem due to the drastic increase of the number of variables and constraints in the formulation. For these reasons, it is required to develop an alternative viable speedup technique that is best suited for the context of my solution with negligible decrease of analysis quality.

It is trivial that all timing paths cannot be critical simultaneously, and not surprisingly, a large part of information on the timing paths is redundant and can be excluded from my formulation. Therefore, I can exploit this property as a means of speeding up. In some aspect, my speedup technique is similar to that in [23] which is a sort of criticality-dependency aware timing analysis. The work in [23] collected timing-risky flip-flops and checked the criticality for all fan-out flip-flops by applying breadth first search traversal. I update the technique in [23] as Algorithm 1 based on my observation.

Algorithm 1 shows my proposed algorithm for speeding up of solving clock period minimization problem Equation (2.5), which classifies the types of all flip-flops and data paths. Meaning of each flip-flop's type is as follows:

- $TYPE_N$-0: It means that there is no need to include the flexible timing property of the flop-flop in Equation (2.5). The maximum clock-to-Q delay of that flip-flop will be set as $\alpha \times t_{c2q}^{min}$ for reduction of the problem size.

- $TYPE_N$-1: It means that the flexible timing property of the flip-flop should be fully or partially considered in Equation (2.5).

The algorithm requires two additional input parameters: a set of maximum allowable clock-to-Q delays t_{c2q}^{upr}, which is related to interacting constraints of setup and hold times, and a user-defined parameter α. I first assume the clock period of the circuit as the minimum possible value T_{clk}^{l}. After that, I calculate $t_{skew}^{h,min}$ for all flip-flops and compute $t_{skew}^{su,cor}$ using them. Note that the minimum hold skew for each flip-flop is independent to the clock period, and $\mathcal{H}$ is the function which finds the corresponding

Algorithm 1: Preprocessing for speeding up of solving Equation (2.5)

input : Timing information of a given circuit
Clock-to-Q delay surface model
Maximum allowable clock-to-Q delays t_{c2q}^{upr}
User-defined parameter α
output: Classification results for all flip-flops

/* initialization */
$T_{clk}^{l} \leftarrow \max_{(i,j)} \left\{ s_0(j) + \left(x(i) + t_{c2q}^{min}(i) + D_{i,j}^{max} \right) - x(j) \right\}$
initialize an empty queue Q

/* rough estimation */
for *each flip-flop* FF_j **do**
  $t_{skew}^{h,min}(j) \leftarrow \min_{FF_i \in P_j} \left\{ x(i) + t_{c2q}^{min}(i) + D_{i,j}^{min} \right\} - x(j)$
  $t_{skew}^{su,cor}(j) \leftarrow \mathcal{H}\left(t_{skew}^{h,min}(j), \alpha \times t_{c2q}^{min}(j) \right)$
  $t_{skew}^{su,wst}(j) \leftarrow \left(T_{clk}^{l} + x(j) \right) - \max_{FF_i \in P_j} \left\{ x(i) + \alpha \times t_{c2q}^{min}(i) + D_{i,j}^{max} \right\}$
  if $t_{skew}^{su,wst}(j) < t_{skew}^{su,cor}(j)$ **then**
    $type_N(j), t_{c2q}^{max}(j) \leftarrow TYPE_N\text{-}1, t_{c2q}^{upr}(j)$
    $Q.enqueue(j)$
  else
    $type_N(j), t_{c2q}^{max}(j) \leftarrow TYPE_N\text{-}0, \alpha \times t_{c2q}^{min}$
  end
end

/* work list algorithm */
while $Q.empty() = False$ **do**
  $i \leftarrow Q.dequeue()$
  for *each flip-flop* FF_j *driven by* FF_i **do**
    if $type_N(j) = TYPE_N\text{-}0$ **then**
      $t_{skew}^{su,wst}(j) \leftarrow \min \left\{ t_{skew}^{su,wst}(j), \left(T_{clk}^{l} + x(j) \right) - \left(x(i) + t_{c2q}^{max}(i) + D_{i,j}^{max} \right) \right\}$
      if $t_{skew}^{su,wst}(j) < t_{skew}^{su,cor}(j)$ **then**
        $type_N(j), t_{c2q}^{max}(j) \leftarrow TYPE_N\text{-}1, t_{c2q}^{upr}(j)$
        $Q.enqueue(j)$
      end
    end
  end
end

return $type_N$

setup skew for a given hold skew and a target clock-to-Q delay on a clock-to-Q delay surface. I then calculate $t_{skew}^{su,wst}$ assuming that the maximum clock-to-Q delays of all precedent flip-flops are equal to $\alpha \times t_{c2q}^{min}$. If $t_{skew}^{su,wst} \geq t_{skew}^{su,cor}$, the minimum hold skew of the flip-flop is large enough, so that there is no need to involve its flexible flip-flop timing property. At last, I propagate the effects of $TYPE_N$-1 flip-flops in the way of work list algorithm using the queue Q.

Connections around a $TYPE_N$-1 flip-flop can be classified into four types by the types of their fan-in and fan-out flip-flops. When it is connected to no $TYPE_N$-1 fan-out flip-flops, it is separated from other flip-flops, and there is no need to include its t_{c2q}^{max}. Hence it is enough to consider only the constraints related to the minimum setup skew and the interacting constraints of the setup and hold times for it. For other cases, it is not guaranteed that t_{c2q}^{max} values of all $TYPE_N$-1 fan-out flip-flops are less than or equal to $\alpha \times t_{c2q}^{min}$, and hence I need to reserve them as optimization variables in Equation (2.5). Remaining flip-flops except $TYPE_N$-1 flip-flops, i.e., $TYPE_N$-0 flip-flops, can be excluded from solving Equation (2.5) and as a result, I could expect the reduction of the number of variables and constraints and total run time.

Algorithm 1 can also be extended to the clock skew scheduling problem with slight modification in the same token. In the case of Equation (2.10), unlike Equation (2.5), x values are included in optimization variables, and therefore, I cannot obtain the exact minimum hold skew for each flip-flop and should handle it as an interval. For example, in the circuit in Figure 2.6, I define n_1 and n_2 as

$$n_1 \coloneqq \underset{n}{\operatorname{argmax}} \left\{ \gamma_2(i,n) + \alpha \times t_{c2q}^{min}(i,n) + D_{(i,n,j)}^{max} \right\},$$

$$n_2 \coloneqq \underset{n}{\operatorname{argmin}} \left\{ \gamma_1(i,n) + t_{c2q}^{min}(i,n) + D_{(i,n,j)}^{min} \right\},$$

in order to analyze and use the maximum clock-to-Q delay of each flip-flop. Then the candidate minimum setup and hold skews of FF_j when $x(j) = 0$ can be expressed as a rectangle whose width and height are $\gamma_2(i,n_1) - \gamma_1(i,n_1)$ and $\gamma_2(i,n_2) - \gamma_1(i,n_2)$, respectively. Since an increment of $x(j)$ causes an increase of $t_{skew}^{su}(j)$ and a decrease of $t_{skew}^{h}(j)$ of the center point of the rectangle simultaneously, the region would shift

if $x(j)$ varies from $\gamma_1(j)$ to $\gamma_2(j)$. Consequently, all candidate pairs of $t_{skew}^{su,min}(j)$ and $t_{skew}^{h,min}(j)$ can be represented as a hexagonal region, and it is enough to check clock-to-Q delays at two vertices of the region due to the convexity of a clock-to-Q delay surface. In Algorithm 1, I can use this concept instead of $t_{skew}^{su,cor}$ and $t_{skew}^{su,wst}$ for classifying the types of all flip-flops.

2.3 Experimental Results

All the experiments were run in MATLAB environments, and my flexible flip-flop timing model based formulations were solved by using CVX, a package for specifying and solving convex programs [88, 89], with SDPT3 4.0, a MATLAB software for semidefinite-quadratic-linear programming [90, 91]. All implementations were performed on Linux machine with 8 cores of 3.50GHz CPU and 16GB memory and targeted on ISCAS'89 benchmark circuits, b19 of ITC'99 benchmark circuits, and des (perf_opt), pci, usb_funct, and vga_lcd of OpenCores benchmark circuits. 45nm NanGate Open Cell Library [4] and SPICE simulations using Synopsys HSPICE were used to sample clock-to-Q delays. Synopsys Design Compiler, IC Compiler, and PrimeTime were used for synthesizing, placement & routing, and extracting timing information of the circuits, respectively.

2.3.1 Application to Minimum Clock Period Finding

To compare the results of the conventional flexible flip-flop timing model based clock period minimization methods proposed in [20] and my clock-to-Q delay interval based analysis (without speedup technique) fairly, I implemented them in MATLAB environments. In addition, I assumed all clock arrival times had already been scheduled under the conventional fixed flip-flop timing model based method [1], since useful clock skews are frequently utilized in circuit design flow for the purpose of optimization. The experimental results are shown in Table 2.1.

Table 2.1: Comparison of minimum clock periods and run times assuming clock arrival times have already been scheduled under the conventional fixed flip-flop timing model [1].

Ckt.	#FFs	Run time [sec]		T_{clk}^{min} [ps]		
		ITA [20]	Proposed	ITA [20]	Proposed	Improvement
s27	3	0.02	0.07	439.75	439.00	0.75
s298	14	0.05	0.08	905.16	904.09	1.08
s344	15	0.06	0.09	909.43	908.37	1.06
s349	15	0.06	0.09	913.94	912.93	1.01
s382	21	0.08	0.09	881.94	880.80	1.14
s386	6	0.01	0.09	903.01	902.96	0.06
s400	21	0.18	0.12	930.18	929.19	0.99
s420	16	0.05	0.09	941.68	938.73	2.95
s444	21	0.08	0.09	919.16	918.27	0.89
s510	6	0.03	0.08	1041.59	1040.46	1.13
s526	21	0.09	0.09	944.33	943.38	0.95
s641	14	0.02	0.09	3259.33	3258.76	0.57
s713	14	0.03	0.09	3456.06	3455.48	0.58
s820	5	0.01	0.08	1316.92	1316.88	0.04
s832	5	0.02	0.07	1246.86	1245.90	0.95
s838	32	0.17	0.10	1029.96	1028.17	1.79
s953	29	0.07	0.10	1197.13	1197.01	0.13
s1196	18	0.02	0.08	1284.22	1283.61	0.61
s1238	18	0.02	0.08	1228.26	1227.51	0.75
s1423	74	0.36	0.17	2983.67	2974.80	8.86
s1488	6	0.01	0.08	1279.83	1279.68	0.14
s5378	162	0.30	0.22	1190.28	1190.20	0.08
s9234	132	0.28	0.22	2439.80	2430.73	9.07
s13207	214	0.38	0.23	2498.22	2489.42	8.80
s15850	128	0.34	0.20	2845.97	2837.20	8.77
s35932	1728	2.25	0.96	1502.32	1501.83	0.48
s38417	1462	2.13	1.60	2130.43	2121.32	9.11
s38584	1159	2.05	0.80	2388.52	2379.63	8.88
b19	1876	6.05	2.01	5999.09	5989.90	0.02
des	1984	2.46	1.29	2293.53	2293.50	0.02
pci	3271	4.16	2.25	2400.64	2400.04	0.60
usb_funct	1735	0.46	0.72	1297.53	1296.86	0.67
vga_lcd	17055	69.47	M/E	3743.12	M/E	N/A

Table 2.1 summarizes minimum clock periods obtained by ITA [20] and the proposed method with run times. The first and second columns, Ckt. and #FFs, represent the name and the number of flip-flops of each benchmark circuit. For some of benchmark circuits, there were discrepancies of minimum clock periods T_{clk}^{min} between ITA and mine. For example, ITA determined that the minimum clock period of s38417 is 2130.43ps; however, my solution reported that it could be reduced further to 2121.32ps. As it can be seen from Table 2.1, all minimum clock periods were reduced by my method. In terms of run time, the proposed method took a little more time than ITA for small circuits, e.g., s27, s386, s420, s641; on the other hand, for large circuits, e.g., s35932, s38584, b19, pci, my method took less time due to its stability nature originated from convexity. Note that M/E in the table represents out-of-memory error.

2.3.2 Application to Clock Skew Scheduling

Experiments of clock skew scheduling for worst slack maximization were started with initializing reference clock periods using conventional fixed flip-flop timing model based clock skew scheduling scheme [1]. For each benchmark circuit, I compared worst slack values under the reference clock period obtained by (1) applying clock skew scheduling utilizing flexible flip-flop timing (CSS-FT) proposed in [2] to the result of [1] additionally, (2) solving my flexible flip-flop timing model based convex problems described in Section 2.2.4 with CVX directly. All clock arrival times were assumed to be in the range from -150ps to +150ps. I also applied the same definitions of setup and hold slacks used in [2] and Section 2.2.4 for calculating them. Results of the experiments are shown in columns 3-5 of Table 2.2. Columns 3 and 4 of the table shows worst slacks of CSS-FT and mine, respectively. As shown in the table, all worst slacks obtained by my method were greater than or equal to those obtained by CSS-FT since I considered it globally while CSS-FT did it locally. Compared with the results of CSS-FT for benchmark circuits, e.g., s386, s838, s953, s1488, my proposed method

Table 2.2: Comparison of worst slacks, minimum clock periods, and total slacks obtained by previous researches and my method.

Ckt.	#FFs	t_{slk}^{w} [ps]			T_{clk}^{min} [ps]			t_{slk}^{tot} [ps]		
		CSS-FT [2]	Proposed	Improvement	CSS [1]	Proposed	Improvement	CSS-FT [2]	Proposed	Improvement
s27	3	73.26	74.34	1.08	503.28	438.96	64.32	382.74	891.44	8.80
s298	14	75.00	75.11	0.11	969.01	903.97	65.04	3156.44	3873.69	717.25
s344	15	75.05	75.05	0.00	973.75	908.37	64.99	3309.81	3748.24	438.42
s349	15	74.97	75.16	0.19	977.87	912.79	65.08	3161.56	3700.39	538.83
s382	21	75.13	75.18	0.04	945.86	880.75	65.10	3032.47	3679.62	647.15
s386	6	19.11	36.71	17.60	922.05	872.85	49.21	563.70	795.68	231.98
s400	21	74.93	75.11	0.18	994.12	929.07	65.04	3619.72	4590.68	970.96
s420	16	75.01	75.10	0.08	1005.68	938.21	67.47	2717.76	3238.24	520.48
s444	21	74.67	75.10	0.43	982.87	917.83	65.04	3589.79	4473.75	883.96
s510	6	75.12	75.12	0.00	1105.51	1040.46	65.05	1040.57	1171.39	130.82
s526	21	74.57	75.09	0.52	1008.08	943.05	65.03	3788.39	4970.82	1182.43
s641	14	16.03	17.48	1.45	3728.24	3245.27	32.97	3810.11	4660.41	850.30
s713	14	16.03	17.48	1.45	3474.97	3442.00	39.55	3422.62	4257.65	835.03
s820	5	16.08	23.39	7.31	1335.84	1296.29	39.55	291.45	458.45	166.00
s832	5	74.46	74.78	0.32	1310.48	1245.77	64.71	557.54	646.78	89.24
s838	32	39.70	66.23	26.53	1093.90	1026.87	67.03	4757.35	5937.16	1179.81
s953	29	34.75	56.12	21.37	1231.80	1166.78	65.92	4723.87	5780.30	1056.43
s1196	18	6.48	17.48	11.00	1303.13	1270.16	32.98	4076.08	5208.91	1132.84
s1238	18	6.50	17.49	10.99	1247.05	1214.06	32.99	3781.89	5038.83	1256.94
s1423	74	75.50	75.51	0.00	3050.64	2975.14	75.50	21544.53	23745.83	2201.30
s1488	6	34.84	55.63	20.79	1314.58	1249.77	64.80	541.63	668.43	126.79
s5378	162	37.27	44.19	6.91	1209.19	1167.32	41.87	36732.06	47125.46	10393.40
s9234	132	75.48	75.54	0.06	2506.72	2431.19	75.53	35781.64	41272.52	5490.88
s13207	214	75.19	75.22	0.04	2565.16	2489.94	75.22	56039.09	64436.94	8397.85
s15850	128	73.93	75.23	1.30	2912.96	2837.72	75.23	31611.01	34006.96	2395.95
s35932	1728	18.90	18.90	0.00	1521.31	1502.31	19.00	559817.35	615888.62	56071.27
s38417	1462	75.60	75.61	0.01	2197.36	2121.75	75.61	391052.57	423355.40	32302.84
s38584	1159	75.31	75.49	0.18	2455.41	2379.93	75.49	334020.99	366855.19	32834.19
b19	1876	75.52	75.52	0.00	6065.95	5990.43	75.52	708851.21	736437.16	27585.95
des	1984	18.99	19.00	0.00	2312.49	2293.49	19.00	400060.43	440894.99	40834.56
pci	3271	19.00	19.00	0.00	2419.56	2400.56	19.00	410575.41	441053.41	30477.99
usb_funct	1735	5.44	17.47	11.03	1316.39	1283.41	32.98	29011.55	38375.92	9365.36
vga_lcd	17055	N/A	M/E	N/A	3762.12	M/E	N/A	N/A	M/E	N/A

ameliorated worst slacks by 10ps$\sim$30ps. In the case of vga_lcd, my method and t_{slk}^{w} estimation process could not be performed due to out-of-memory errors originated from the size of the circuit.

The worst slack value of a circuit linked directly to the minimum clock period of it. Basically, a minimum clock period means that there are no timing violations at all in the circuit under that timing condition, i.e. $t_{slk}^{w} \geq 0$; therefore, comparing minimum clock periods obtained by changing clock arrival times can also be one criterion for assessing the performance of clock skew scheduling, so I proceeded the experiments additionally. Since [20] cannot modify clock arrival times and [2] only works under given clock period with initial clock arrival times, both of them cannot be compared with mine directly; thus I estimated the differences between minimum clock periods obtained by fixed flip-flop timing model based clock skew scheduling scheme [1] and my method. Note that the results of [1] are optimal under the conventional fixed flip-flop timing model. Results are shown in columns 6-8 of Table 2.2. The table shows that my flexible flip-flop timing model based clock skew scheduling reduced minimum clock periods by up to 75ps over the fixed flip-flop timing model based approach.

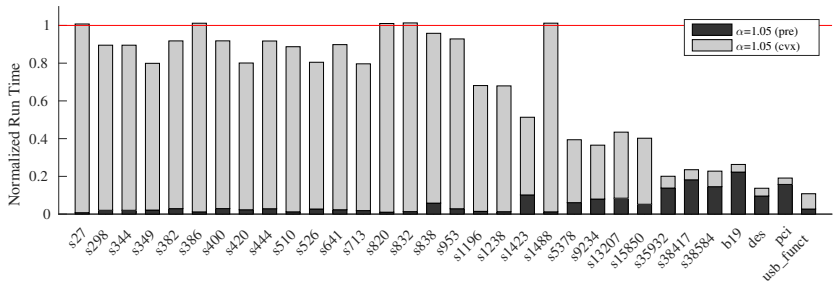
Columns 9-11 of Table 2.2 summarizes the total slacks of CSS-FT and my approach. For all benchmark circuits, mine improved total slacks by up to 56ns over those of CSS-FT. While my method is effective in comparison with other conventional methods, complexity or run times of them would be one of the most critical issues. For example, CSS-FT with conventional fixed flip-flop timing model based clock skew scheduling took $0.87 + 3.10 = 3.97$ seconds for relaxing timing constraints of pci in Table 2.3, but mine took 21.85 seconds for resolving them, without any speedup techniques, which is about 5.5$\times$ slower. In addition, for all kinds of experiments, my method could not handle vga_lcd due to its size. Thus it is needed to refine my solution to overcome them by exploiting scalable speedup technique.

Table 2.3: Comparison of run times of CSS [1], CSS-FT [2], and my method for maximizing worst slack for some benchmark circuits. Note that run times of CSS-FT denote additional processing times only.

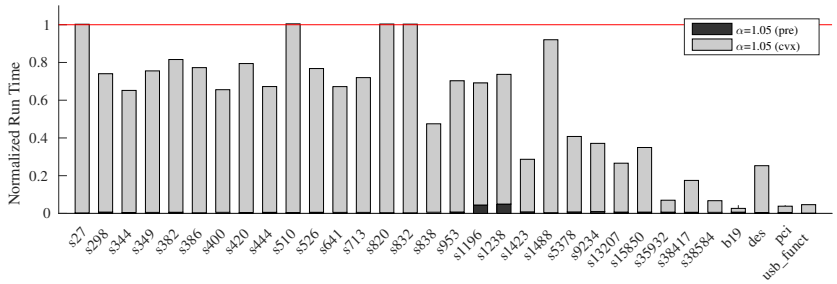
Ckt.	#FFs	Average run time [sec]		
		CSS [1]	CSS-FT [2]	Proposed
s27	3	0.05	0.02	0.10
s349	15	0.06	0.04	0.17
s386	6	0.05	0.05	0.12
s444	21	0.08	0.06	0.23
s526	21	0.06	0.04	0.19
s641	14	0.05	0.03	0.14
s713	14	0.08	0.07	0.15
s820	5	0.05	0.04	0.12
s832	5	0.05	0.04	0.12
s838	32	0.08	0.13	0.30
s953	29	0.06	0.06	0.23
s1196	18	0.06	0.05	0.14
s1238	18	0.07	0.04	0.15
s1423	74	0.11	0.22	0.61
s1488	6	0.05	0.02	0.11
s5378	162	0.11	0.20	0.72
s9234	132	0.09	0.21	0.77
s13207	214	0.12	0.29	0.96
s15850	128	0.07	0.16	0.69
s35932	1728	0.19	1.48	7.36
s38417	1462	0.65	2.09	11.80
s38584	1159	0.31	1.20	6.99
b19	1876	0.86	3.27	14.81
des	1984	0.44	1.35	12.08
pci	3271	0.87	3.10	21.85
usb_funct	1735	0.10	0.43	6.36

2.3.3 Efficacy of Scalable Speedup Technique

For verifying the effectiveness of my speedup technique presented in Section 2.2.5, I compared analysis qualities and run times obtained with and without the technique. I set the same experimental environments used in Sections 2.3.1 and 2.3.2. For speeding up of finding solutions, I set $\alpha = 1.05$, i.e., I excluded some variables and constraints of the flip-flops whose maximum clock-to-Q delay is less than $1.05 \times t_{c2q}^{min}$ from my formulations. The results are shown in Figure 2.7.



(a) Comparison of run times of finding minimum clock periods.



(b) Comparison of run times of clock skew scheduling for maximizing worst slack with clock arrival times in the range from -50ps to +50ps.

Figure 2.7: Effectiveness of my speedup technique for the problems. $\alpha = 1.05$ is used for speeding up and all run times are normalized by run times of mine without the speedup technique. *pre* and *cvx* in the charts represent preprocessing and solving with convex optimization solver, respectively, and target clock periods for (b) was obtained by conventional fixed flip-flop timing model based clock skew scheduling scheme [1].

Figure 2.7 shows the normalized run times of my speeding up solutions for the problem of finding minimum clock period and the clock skew scheduling problem for worst slack maximization, respectively. All run times are normalized by the run times of my original solutions; therefore, the value of 0.10 of usb_funct in Figure 2.7(a), for example, means that I reduced the run time by 90%. As it can be seen from the figure, run times were reduced by 33% and 44% on average. Run times for small circuits like s27 and s832 were slightly increased due to the preprocessing step; on the other hand, for large size circuits, they were especially reduced considerably since the number of critical flip-flops and data paths were not proportional to the size of a given circuit. For instance, they were reduced by 74% and 97% for analyzing and optimizing b19, respectively. Furthermore, the problems of vga_lcd were solved within 4.81 seconds and 12.22 seconds, which are 14.44× and 5.11× faster than those of ITA and CSS-FT, respectively, with no out-of-memory errors. Meanwhile, the averages of the differences of the results obtained with and without proposed speedup technique were only 1.20ps and 1.40ps, respectively. Therefore, it can be concluded that my technique can efficiently remove unnecessary variables and constraints from my original formulations.

2.4 Summary

In this chapter, I proposed a new clock-to-Q delay interval analysis and developed its mathematical formulation for applicability to representative timing analysis and optimization problems, i.e., finding minimum clock period and clock skew scheduling, with the scalable speedup technique. Experimental results with benchmark circuits demonstrated improvement of optimization quality in terms of clock period, worst slack, and total slack, as well as discrepancies between the results from previous works and my proposed concepts. Furthermore, it was shown that my speedup technique can significantly shorten run times of solving the problems with very little loss of optimality.

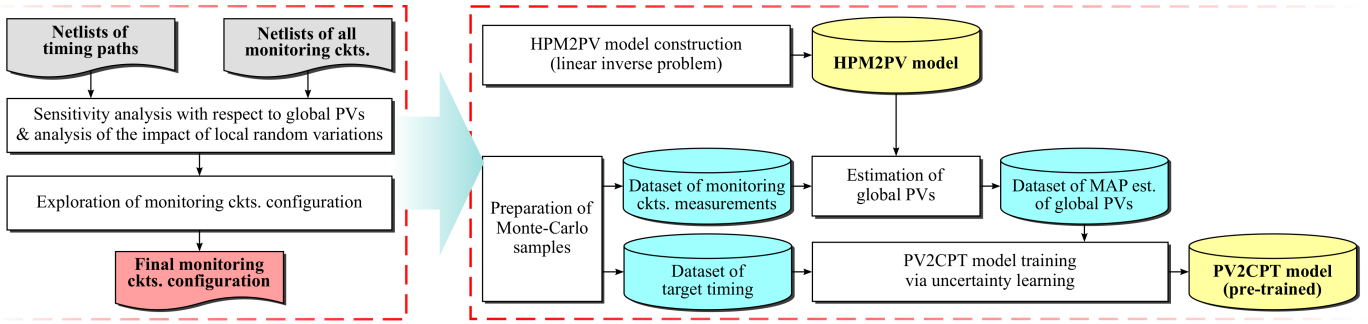
Chapter 3

HARDWARE PERFORMANCE MONITORING METHODOLOGY AT NTC AND ADVANCED TECHNOLOGY NODE

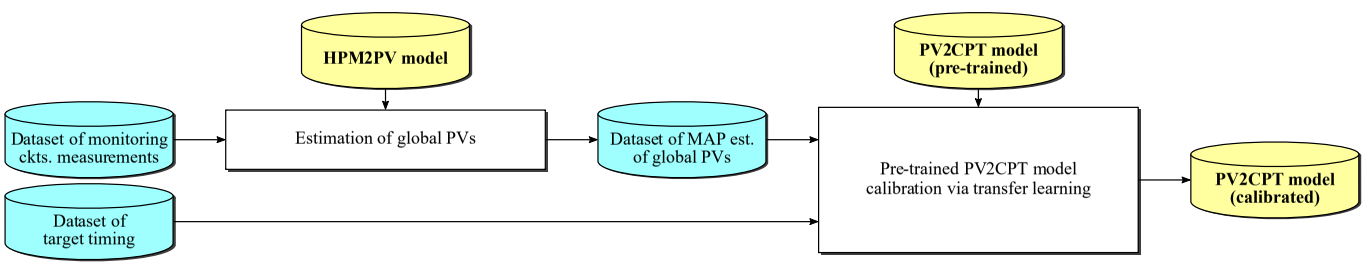
3.1 Overall Flow of Proposed HPM Methodology

Figure 3.1 shows the overall flow of my HPM methodology. According to typical IC design flow, I consider the procedures in a design phase (Figure 3.1(a)), silicon characterization step (Figure 3.1(b)), and volume production phase (Figure 3.1(c)) separately in my HPM methodology. Before constructing a target timing prediction model, I generate statistical netlists of monitoring circuits and timing paths related to the target timing through modeling and characterization of BEOL PVs (Section 3.2.1). Then I prepare small datasets consisting of SPICE simulation results of the circuits and start to train their surrogate models (Section 3.2.2), which will replace remaining SPICE simulations during the design phase. Note that I use the statistical netlists in the SPICE simulations for including the effect of FEOL and BEOL PVs on measurements of monitoring circuits and a target timing simultaneously.

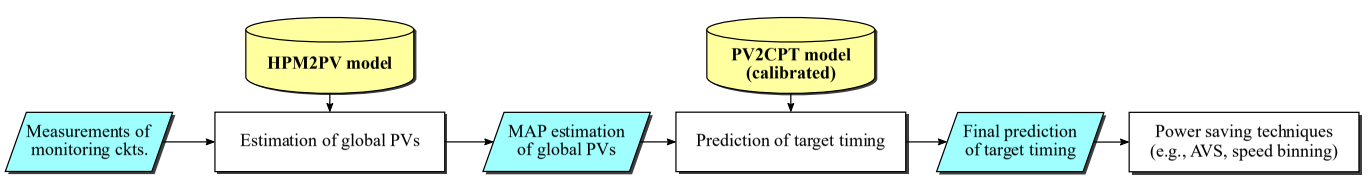
The quality of a target timing prediction significantly depends on the configuration of monitoring circuits, i.e., which one will be used as a monitoring circuit and



(a) Design phase



(b) Silicon characterization step



(c) Volume production phase

Figure 3.1: The overall flow of my proposed HPM methodology.

how many of its instances will be measured. Therefore, I first optimize the configuration of monitoring circuits considering design dependency and the expected prediction pessimism (Section 3.3.2), as illustrated in the left side in Figure 3.1(a). From the optimized configuration, I construct the following two models (right side in Figure 3.1(a)).

- HPM2PV (Section 3.3.1): A model for estimating FEOL and BEOL PVs from measurement results of the set of monitoring circuits.
- PV2CPT (Section 3.3.3): A model for predicting variation of a target timing considering prediction yield from estimation results of the HPM2PV model.

In spite of substantial care, there might be a gap between prediction results through the trained model and actual results measured from fabricated chips due to various kinds of factors. To resolve this, as shown in Figure 3.1(b), I finely tune the parameters of the pre-trained PV2CPT model using measurement data in a silicon characterization step (Section 3.4.1). In a volume production phase (Figure 3.1(c)), I sequentially estimate PVs from measurements of the monitoring circuits of a chip using the HPM2PV model and predict the target timing through fine-tuned PV2CPT model at last (Section 3.4.2).

3.2 Prerequisites to HPM Methodology

3.2.1 BEOL Process Variation Modeling

In the proposed HPM methodology, I consider not only FEOL PVs but also BEOL PVs in a comprehensive way to reflect growing performance variation caused by BEOL PVs in advanced process nodes [92]. Unlike FEOL PVs, statistical models related to BEOL PVs are not provided in general [93, 94], which are necessary for optimizing the configuration of monitoring circuits and constructing prediction models. Thus I generate statistical models of BEOL PVs first. Precisely, I consider metal resistances and ground capacitances of all interconnect layers and via resistances between every two adjacent interconnect layers as the components of BEOL PVs. For example, I as-

sume two random variables $x_{r,i}$∼N$(0, \sigma_{r,i}^2)$ and $x_{c,i}$∼N$(0, \sigma_{c,i}^2)$ to create the models of metal resistance and ground capacitance of interconnect layer i. Likewise, I introduce a random variable $x_{v,i}$∼N$(0, \sigma_{v,i}^2)$ for expressing the variation model of a via resistance between interconnect layers i and $i+1$.

Figure 3.2 shows four groups of metal resistances and parasitic capacitances to which the models of BEOL PVs will be applied. I express the variations of metal resistance R'_r and ground capacitance C'_g of layer i with their nominal values (R_r and C_g) and some related coefficients ($k_{r,i}$ and $k_{c,i}$) as follows.

$$R'_r = R_r \times (1 + k_{r,i} \times x_{r,i})$$
$$C'_g = C_g \times (1 + k_{c,i} \times x_{c,i})$$

R_r and C_g can be found in parasitic extraction (PEX) results at the nominal condition. In addition, by assuming $x_{r,i}$ and $x_{c,i}$ as $3\sigma_{r,i}$ and $3\sigma_{c,i}$ at Cmin and Cmax corners respectively, I can derive $k_{r,i}$ and $k_{c,i}$ from the PEX results at those corners. In the same way, I can calculate the value of $k_{v,i}$ related to a via resistance between interconnect layers i and $i+1$, from which the variation model of via resistance R'_v can be expressed with its nominal value R_v as follows.

$$R'_v = R_v \times (1 + k_{v,i} \times x_{v,i})$$

Meanwhile, $x_{r,i}$ and $x_{c,i}$ are not mutually independent for each interconnect layer i. Hence I need to characterize correlation ρ_i between them, of which the value can be inferred from the PEX results at the five BEOL corners, i.e., Cmax, Cmin, RCmax, RCmin, and nominal. In my methodology, from multivariate distributions of R'_r and C'_g, I characterize ρ_i by assuming that the Mahalanobis distance between the middle of $(R'_r, C'_g)_{\text{Cmax}}$ and $(R'_r, C'_g)_{\text{RCmax}}$ and the distribution center (R_r, C_g) is 3. Furthermore, I generate the variation model of coupling capacitance C'_c formed between two interconnect metal segments which belong to interconnect layers i and j, respectively, with its nominal value C_c and the linear combination of the related variables (i.e., $x_{r,i}$,

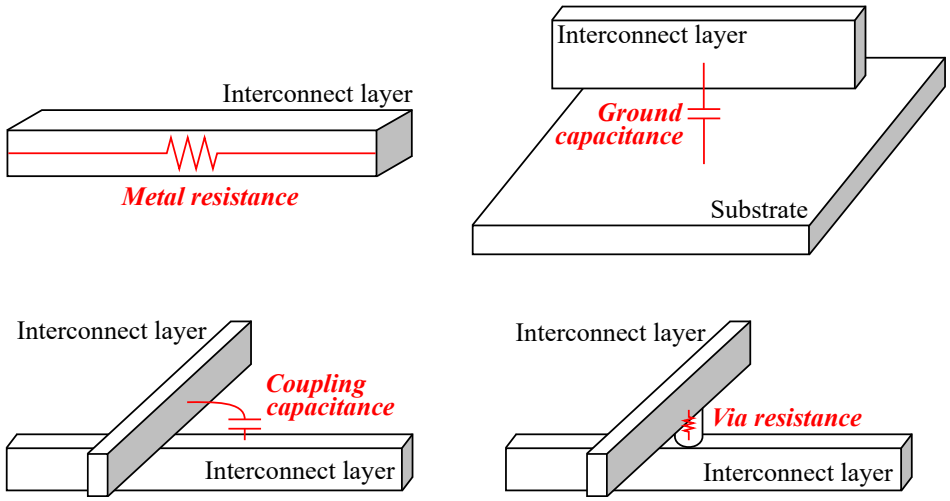


Figure 3.2: Four groups of interconnect resistances and parasitic capacitances considered in my BEOL PVs modeling.

$x_{c,i}$, $x_{r,j}$, and $x_{c,j}$), as follows.

$$C_c' = C_c \times (1 + \alpha_{i,j} \times x_{r,i} + \beta_{i,j} \times x_{c,i} + \gamma_{i,j} \times x_{r,j} + \delta_{i,j} \times x_{c,j})$$

Note that I obtain the values of the coefficients $\alpha_{i,j}$, $\beta_{i,j}$, $\gamma_{i,j}$, and $\delta_{i,j}$ from C_c' values in the PEX results at the five BEOL corners through linear regression. Finally, for an arbitrary circuit netlist, I convert the PEX result at the nominal corner to the statistical netlist that expresses each BEOL component in terms of BEOL PVs, as shown in Figure 3.3.

3.2.2 Surrogate Model Preparation

Since most characterization in my HPM methodology heavily relies on lots of SPICE simulations, there can be an issue related to computing resources, tool licenses, and simulation runtime. To save the usage of SPICE simulations, I exploit the surrogate model of variation behavior for each circuit, of which accuracy is almost the same level as that of SPICE simulation while the evaluation is much faster. Stochastic spec-

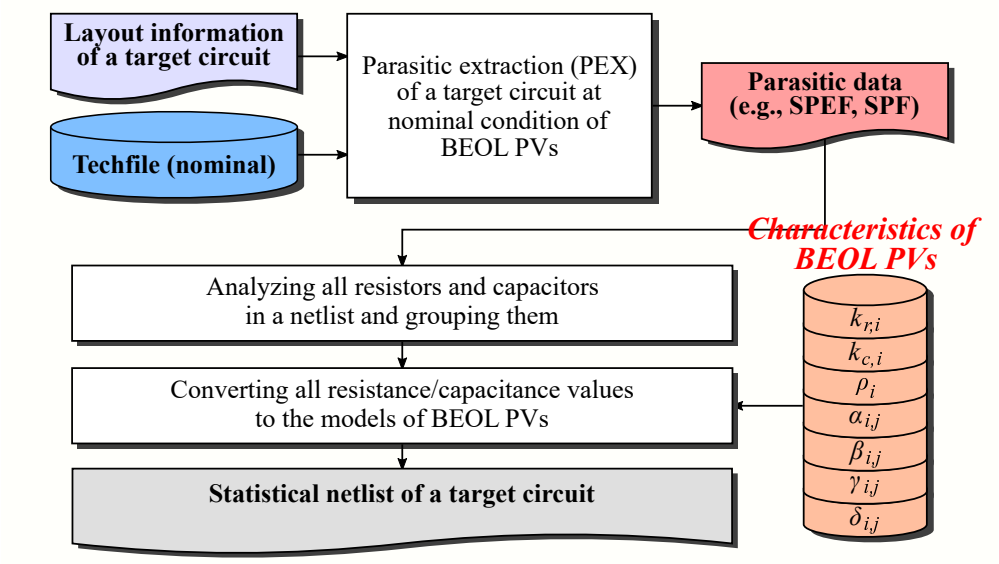


Figure 3.3: Conversion flow of a statistical netlist considering BEOL PVs.

tral method [95, 96] is one of the most widely used surrogate model based stochastic analyses, which implements variation models in the form of the linear combination of some specific basis functions. For example, let me assume that the vector consisting of mutually independent random parameters $\boldsymbol{\xi} = [\xi_1, \cdots, \xi_d] \in \mathbb{R}^d$ and restricted and smooth change of a target output $y(\boldsymbol{\xi})$. Then I can approximate $y(\boldsymbol{\xi})$ using orthonormal basis function set $\{\Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi})\}$ with generalized polynomial chaos expansion (PCE) [97] as

$$y(\boldsymbol{\xi}) \approx \sum_{|\boldsymbol{\alpha}|=0}^{p} c_{\boldsymbol{\alpha}} \Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi}),$$

where $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_d] \in \mathbb{N}^d$ indicates the highest polynomial order of each parameter in the corresponding basis, and the polynomial order $|\boldsymbol{\alpha}| = |\alpha_1| + \cdots + |\alpha_d|$ is bounded by p.

I can obtain $c_{\boldsymbol{\alpha}}$ in the equation from the projection framework [98], but a proper numerical technique is required to calculate the integration in it. If I define $\{(\xi_k^{i_k}, w_k^{i_k})\}_{i_k=1}^{n}$ as the set of pairs consisting of 1-D quadrature samples and weights that correspond

to ξ_k, I can express $c_{\boldsymbol{\alpha}}$ as follows.

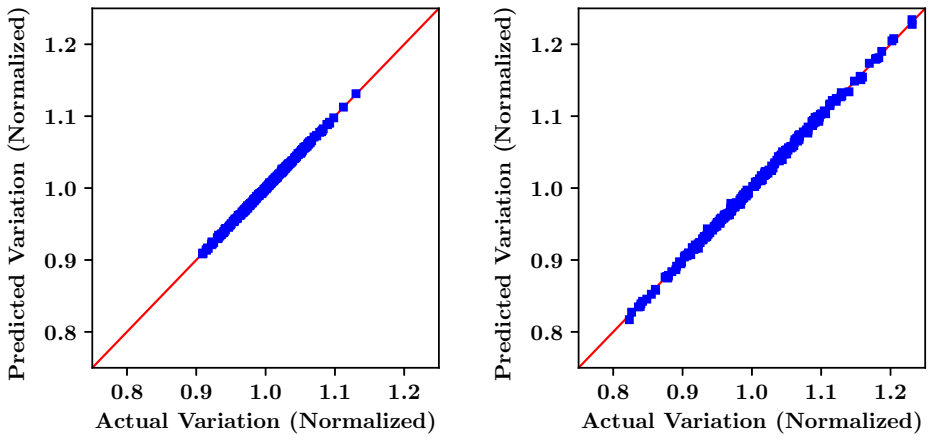
$$c_{\boldsymbol{\alpha}} = \sum_{1 \leq i_1, \cdots, i_d \leq n} y(\boldsymbol{\xi}_{i_1,\cdots,i_d}) \Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi}_{i_1,\cdots,i_d}) w_{i_1,\cdots,i_d} = \langle \boldsymbol{\mathcal{Y}}, \boldsymbol{\mathcal{W}}_{\boldsymbol{\alpha}} \rangle$$

Note that $\boldsymbol{\xi}_{i_1,\cdots,i_d} = [\xi_1^{i_1}, \cdots, \xi_d^{i_d}]$ and $w_{i_1,\cdots,i_d} = w_1^{i_1} \cdots w_d^{i_d}$ are a multidimensional quadrature sample and weight, respectively, and $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and $\boldsymbol{\mathcal{W}}_{\boldsymbol{\alpha}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ denote the tensors of which elements indexed by $(i_1, \cdots, i_d)$ are $y(\boldsymbol{\xi}_{i_1,\cdots,i_d})$ and $\Psi_{\boldsymbol{\alpha}}(\boldsymbol{\xi}_{i_1,\cdots,i_d}) w_{i_1,\cdots,i_d}$, respectively. As a result, I need to run the simulation n^d times for directly computing $\boldsymbol{\mathcal{Y}}$, which is impractical for a high-dimensional case. To mitigate this difficulty, Zhang, Weng, and Daniel [99] proposed the method of generating the tensor $\boldsymbol{\mathcal{X}}$ that approximates the original tensor $\boldsymbol{\mathcal{Y}}$ efficiently given small subset Ω of the index set $\mathcal{I}$ that consists of all indexes of $\boldsymbol{\mathcal{Y}}$. In particular, they added two constraints for relaxing ill-posedness of the problem as follows.

1. Low-rank constraint: It is expected that the approximation $\boldsymbol{\mathcal{X}}$ of the original tensor $\boldsymbol{\mathcal{Y}}$ has a low-rank decomposition.

2. Sparse constraint: The ℓ_1-norm of a vector collecting all coefficients $c_{\boldsymbol{\alpha}}$ should be very small.

Figures 3.4(a) and (b) show the accuracy of the surrogate models for the variation of a ring oscillator and timing path for 28nm process technology, respectively. The x- and y-axis denote timing variations obtained from SPICE simulations and predicted through the surrogate models. The sizes of training and validation datasets were 500 and 250, respectively, and I tested with 250 samples for checking their accuracy. The total number of process parameters in the surrogate model training was 38 (19 FEOL parameters and 20 BEOL parameters). Through comparison, I observe that the surrogate models are very accurate, of which errors are -0.03% for ring oscillators and -0.02% for timing paths, on average, respectively.



(a) Ring oscillator

(b) Timing path

Figure 3.4: Comparison between actual (from SPICE simulations) and predicted (through surrogate models) timing variations for (a) one ring oscillator and (b) one timing path. Blue squares and red lines indicate test results and the ideal prediction (i.e., $y = x$), respectively.

3.3 HPM Methodology: Design Phase

3.3.1 HPM2PV Model Construction

The fundamental assumption of my HPM2PV model is that the measurements of monitoring circuits can be represented as the linear combination of global PVs as

$$\widetilde{d}_i = \overline{d}_i + \sum_j \xi_{ji} x_j + \varepsilon_i.$$

$\widetilde{d}_i$ and $\overline{d}_i$ denote the measurement result of the i-th monitoring circuit and its expectation, respectively, and ξ_{ji} indicates the sensitivity of the i-th monitoring circuit measurement to the variation of the j-th process parameter. x_j and ε_i are the j-th PV and a random variation component included in the i-th monitoring circuit measurement, respectively. I can also generalize it to the matrix form as

$$\widetilde{\mathbf{d}} = \overline{\mathbf{d}} + \Xi^{\mathrm{T}}\mathbf{x} + \boldsymbol{\varepsilon}. \tag{3.1}$$

Note that I assume that all monitoring circuits are measured in the V_{dd} regime guaranteeing linearity to PVs.

In short, HPM2PV model construction is a linear inverse problem of which the objective is to find PVs $\mathbf{x}$ of a chip from the measurements $\overline{\mathbf{d}}$ of monitoring circuits. However, it is difficult to estimate $\mathbf{x}$ precisely due to an insufficient number of observations and inclusion of purely random components $\boldsymbol{\varepsilon}$ in Equation (3.1). Hence, it is more reasonable to infer the distribution of $\mathbf{x}$ through Bayesian interpretation [100]. Let me assume that for M process parameters and N measurements of monitoring circuits, $\mathbf{x}$ and $\boldsymbol{\varepsilon}$ follow multivariate normal distributions $\mathrm{N}(\mathbf{0}, \Sigma_{\mathbf{x}})$ and $\mathrm{N}(\mathbf{0}, \Sigma_{\boldsymbol{\varepsilon}})$, respectively, where $\Sigma_{\mathbf{x}}$ and $\Sigma_{\boldsymbol{\varepsilon}}$ are their covariance matrices. Then probability density function (pdf) of the prior distribution of $\mathbf{x}$ is

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^M \det\Sigma_{\mathbf{x}}}} \exp\left[-\frac{1}{2}\mathbf{x}^{\mathrm{T}}\Sigma_{\mathbf{x}}^{-1}\mathbf{x}\right].$$

The likelihood of observation $\widetilde{\mathbf{d}}$, i.e., probability of observing $\widetilde{\mathbf{d}}$ given $\mathbf{x}$, also follows a multivariate normal distribution by Equation (3.1), of which pdf is as follows.

$$p(\widetilde{\mathbf{d}}|\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N \det\Sigma_{\boldsymbol{\varepsilon}}}} \exp\left[-\frac{1}{2}\left\{\widetilde{\mathbf{d}} - (\overline{\mathbf{d}} + \Xi^{\mathrm{T}}\mathbf{x})\right\}^{\mathrm{T}}\Sigma_{\boldsymbol{\varepsilon}}^{-1}\left\{\widetilde{\mathbf{d}} - (\overline{\mathbf{d}} + \Xi^{\mathrm{T}}\mathbf{x})\right\}\right]$$

According to Bayes' theorem, the posterior distribution of $\mathbf{x}$ given $\widetilde{\mathbf{d}}$ is proportional to the multiplication of the prior $p(\mathbf{x})$ and the likelihood $p(\widetilde{\mathbf{d}}|\mathbf{x})$, so it follows $\mathrm{N}(\overline{\boldsymbol{\mu}}_{\mathbf{x}}, \overline{\Sigma}_{\mathbf{x}})$ where

$$\overline{\boldsymbol{\mu}}_{\mathbf{x}} = \overline{\Sigma}_{\mathbf{x}}\Xi\Sigma_{\boldsymbol{\varepsilon}}^{-1}(\widetilde{\mathbf{d}} - \overline{\mathbf{d}}), \tag{3.2a}$$

$$\overline{\Sigma}_{\mathbf{x}}^{-1} = \Sigma_{\mathbf{x}}^{-1} + \Xi\Sigma_{\boldsymbol{\varepsilon}}^{-1}\Xi^{\mathrm{T}}. \tag{3.2b}$$

Equation (3.2a) indicates the maximum a posterior (MAP) estimation of $\mathbf{x}$, which is linearly dependent on $\widetilde{\mathbf{d}}$. Equation (3.2b) expresses the estimation uncertainty of $\mathbf{x}$. Thus, it should be taken into account for the pessimistic inference of PVs and subsequent target timing prediction.

3.3.2 Optimization of Monitoring Circuits Configuration

Monitoring circuits configuration, i.e., which type of circuits will be monitored and how many instances will be measured while testing a chip, can dramatically change the estimation of $\mathbf{x}$ and subsequently make the difference in the amount of prediction pessimism. When there exist only two process parameters x_1 and x_2, the prior distribution of $\mathbf{x} = [x_1, x_2]^{\mathrm{T}}$, described in the middle in Figure 3.5, can be transformed to the posterior distributions on the left and right sides in Figure 3.5, depending on the configuration of monitoring circuits. (Note that for the sake of simplicity, I assume that $\overline{\boldsymbol{\mu}}_{\mathbf{x}}$ for both cases is the same as $\mathbf{0}$.) If the contour of target timing is represented as the blue lines in Figure 3.5, I can find out its point of tangency to the confidence region formed by the posterior distribution analytically. In addition, if target timing increases or decreases linearly to PVs, the orthogonal distance from the center of the posterior distribution to its tangential contour will be proportional to the average prediction error. Given the gradient vector of contour $\mathbf{k}$ (black arrow in Figure 3.5), the distance is in proportion to $\sqrt{\mathbf{k}^{\mathrm{T}}\overline{\Sigma}_{\mathbf{x}}\mathbf{k}}$. Therefore, I can consider the following optimization problem.

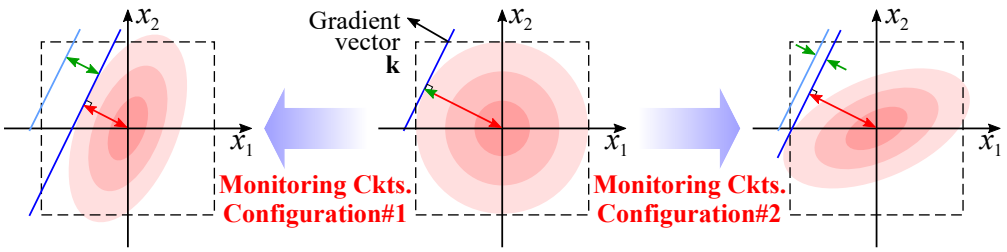


Figure 3.5: Prior (middle) and posterior (left and right) distributions of PVs $\mathbf{x} = [x_1, x_2]^{\mathrm{T}}$. The shape of a posterior distribution can be different depending on the configuration of monitoring circuits. Blue lines represent the contour of target timing (i.e., all points on the same blue line cause the same amount of target timing variation), and red and green arrows indicate the average prediction errors and their decrement when I exploit HPM methodology.

(*Monitoring Circuits Optimization Problem)*: Given T types of monitoring circuits, the total number of their instances N, their sensitivities to PVs, mean and standard deviation of the measurements, and direction of the increase of target timing $\mathbf{k}$, find the configuration of monitoring circuits that derives the most accurate HPM2PV model, i.e., the combination that minimizes $\mathbf{k}^{\mathrm{T}}\overline{\Sigma}_{\mathbf{x}}\mathbf{k}$.

Monitoring Circuits Optimization Problem is analogous to the optimal experiment design (OED) problem [101], which is about finding the best combination of experiments for estimating the underlying parameters accurately with the limited number of observations. Recently, Sagnol and Harman [102] transformed the OED problem of a general form, i.e.,

$$\max_{\mathbf{w}\in\mathcal{W}} \Phi_{\mathrm{D|K}}(\mathrm{M}(\mathbf{w})) = \Phi_{\mathrm{D|K}}\left(\sum_{i=1}^{s} w_i \mathrm{A}_i \mathrm{A}_i^{\mathrm{T}}\right) \tag{3.3}$$

where $\Phi_{\mathrm{D|K}} : \mathrm{M} \rightarrow (\mathrm{detK}^{\mathrm{T}}\mathrm{M}^{-1}\mathrm{K})^{-1/k}$, into a mixed integer second-order cone programming (MISOCP). Note that $\mathcal{W}$ denotes the set of all feasible experiment designs, and $\mathrm{A}_i \in \mathbb{R}^{m\times l_i}$ for $i = 1, \cdots, s$ are known matrices. They also assumed the parameter subsystem $\mathfrak{\vartheta} = \mathrm{K}^{\mathrm{T}}\boldsymbol{\theta}$, $\mathrm{K} \in \mathbb{R}^{m\times k}$, $k \leq m$ with range$(\mathrm{K}) \subseteq$ range(M). Meanwhile, by Equation (3.2b), I can convert the objective of *Monitoring Circuits Optimization Problem* sequentially as follows.

$$\begin{aligned} \min \mathbf{k}^{\mathrm{T}}\overline{\Sigma}_{\mathbf{x}}\mathbf{k} &\Leftrightarrow \min \mathrm{det}\mathbf{k}^{\mathrm{T}}(\Sigma_{\mathbf{x}}^{-1} + \Xi\Sigma_{\boldsymbol{\varepsilon}}^{-1}\Xi^{\mathrm{T}})^{-1}\mathbf{k} \\ &\Leftrightarrow \max \left\{\mathrm{det}\mathbf{k}^{\mathrm{T}}(\Sigma_{\mathbf{x}}^{-1} + \Xi\Sigma_{\boldsymbol{\varepsilon}}^{-1}\Xi^{\mathrm{T}})^{-1}\mathbf{k}\right\}^{-1} \end{aligned} \tag{3.4}$$

From structural similarity between $\mathrm{M}(\mathbf{w}) = \sum_{i=1}^{s} w_i \mathrm{A}_i \mathrm{A}_i^{\mathrm{T}}$ in Equation (3.3) and $\Sigma_{\mathbf{x}}^{-1} + \Xi\Sigma_{\boldsymbol{\varepsilon}}^{-1}\Xi^{\mathrm{T}} = \mathrm{Q}_{\mathbf{x}}\mathrm{Q}_{\mathbf{x}}^{\mathrm{T}} + \sum_{t=1}^{T} n_t(\frac{\boldsymbol{\xi}_t}{\sigma_t})(\frac{\boldsymbol{\xi}_t^{\mathrm{T}}}{\sigma_t})$ in Equation (3.4) where $\mathrm{Q}_{\mathbf{x}}$ is the Cholesky decomposition of $\Sigma_{\mathbf{x}}$, I can write MISOCP formulation of *Monitoring Circuits Optimization Problem* by modifying that in [102]. Upon the observation of Chan, Gupta, Kahng, and Lai [47], I can extend my formulation to optimize the average of

the metrics associated with each cluster of timing paths, i.e.,

$$
\begin{aligned}
\text{maximize} \quad & \frac{1}{W}\sum_{w} J_w \\
\text{subject to} \quad & \mathrm{Q}_{\mathbf{x}}\mathbf{z}_{0,w} + \sum_{t}\frac{\boldsymbol{\xi}_t}{\sigma_t} z_{t,w} = \mathbf{k}_w J_w, && \forall w, \\
& {z_{t,w}}^2 \leq q_{t,w}\frac{n_t}{N+1}, && \forall t, \forall w, \\
& \|\mathbf{z}_{0,w}\|^2 \leq q_{0,w}\frac{1}{N+1}, && \forall w, \\
& q_{0,w} + \sum_{t} q_{t,w} \leq J_w, && \forall w, \\
& q_{t,w} \geq 0, && \forall t, \forall w, \\
& q_{0,w} \geq 0, && \forall w, \\
& \sum_{t} n_t \leq N,
\end{aligned}
\tag{3.5}
$$

where W denotes the number of clusters and $\mathbf{k}_w$ is the direction of target timing increase for the w-th cluster. Note that $\mathbf{z}_{0,w}$, $q_{0,w}$, J_w, $z_{t,w}$, and $q_{t,w}$ are intermediate variables, and $\mathbf{n} = [n_1, \cdots, n_T]^{\mathrm{T}}$ means the target variables I aim to find. For example, n_1 represents the number of instances of the first type monitoring circuit.

Despite its mathematical rigor, Equation (3.5) lacks practicality due to the branch-and-bound based search algorithms for solving it. For improving the scalability, I introduce the tabu search heuristic algorithm suggested by Harman, Bachratá, and Filová [103]. Unlike conventional hill climbing based local search heuristic algorithms [104], it finds the (near-)optimal solution by increasing or decreasing the element of current experimental design to avoid being trapped at a local optimum. Though it includes an update of the objective value (in my case, $\mathbf{k}^{\mathrm{T}}(\Sigma_{\mathbf{x}}^{-1} + \Xi\Sigma_{\boldsymbol{\varepsilon}}^{-1/2}\mathrm{diag}(\mathbf{n})\Sigma_{\boldsymbol{\varepsilon}}^{-1/2}\Xi^{\mathrm{T}})^{-1}\mathbf{k}$) for the solution met during optimization, I can compute it incrementally by exploiting

that $\mathbf{n} = \mathbf{n}_0 \pm \mathbf{e}_i$, $i \in \{1, \cdots, N\}$, where $\mathbf{n}_0$ denotes the previous design, as follows.

$$\begin{aligned}
\mathrm{Z} &= \Sigma_{\mathbf{x}}^{-1} + \mathrm{F}\mathrm{diag}(\mathbf{n})\mathrm{F}^{\mathrm{T}} \\
&= \Sigma_{\mathbf{x}}^{-1} + \mathrm{F}\mathrm{diag}(\mathbf{n}_0 \pm \mathbf{e}_i)\mathrm{F}^{\mathrm{T}} \\
&= \Sigma_{\mathbf{x}}^{-1} + \mathrm{F}\{\mathrm{diag}(\mathbf{n}_0) \pm \mathrm{diag}(\mathbf{e}_i)\}\mathrm{F}^{\mathrm{T}} \\
&= \Sigma_{\mathbf{x}}^{-1} + \mathrm{F}\mathrm{diag}(\mathbf{n}_0)\mathrm{F}^{\mathrm{T}} \pm \mathrm{F}\mathrm{diag}(\mathbf{e}_i)\mathrm{F}^{\mathrm{T}} \\
&= \mathrm{Z}_0 \pm \mathbf{f}_i\mathbf{f}_i^{\mathrm{T}}
\end{aligned}$$

Note that Z_0 and Z are $\overline{\Sigma}_{\mathbf{x}}^{-1}$ before and after updating $\mathbf{n}_0$ to $\mathbf{n}$, respectively, and $\mathrm{F} = \Xi\Sigma_{\boldsymbol{\varepsilon}}^{-1/2} = [\mathbf{f}_1, \cdots, \mathbf{f}_N]$. Since Z_0 and Z_0^{-1} are already known in the previous update, I can calculate Z^{-1} by Sherman-Morrison formula as

$$\mathrm{Z}^{-1} = \left(\mathrm{Z}_0 \pm \mathbf{f}_i\mathbf{f}_i^{\mathrm{T}}\right)^{-1} = \mathrm{Z}_0^{-1} \mp \frac{\mathrm{Z}_0^{-1}\mathbf{f}_i\mathbf{f}_i^{\mathrm{T}}\mathrm{Z}_0^{-1}}{1 \pm \mathbf{f}_i^{\mathrm{T}}\mathrm{Z}_0^{-1}\mathbf{f}_i}.$$

For a general SoC design, I can adjust the total number of monitoring circuit instances. For example, when a more precise prediction is required, it is needed to insert a large number of monitoring circuits. Otherwise, it will be efficient to allocate the minimum number of them for reducing test cost and area overhead. In my methodology, I propose the flow shown in Figure 3.6 to explore this trade-off accurately by exploiting an HPM2PV model and the solver of *Monitoring Circuits Optimization Problem*. For a given prediction error e and the initial number of monitoring circuit instances n_0, I first set the total number N to n_0 and solve the problem. From the optimization result, I construct the corresponding HPM2PV model and derive the estimation uncertainty using Equation (3.2b). Then I run a Monte-Carlo simulation using the surrogate models of the timing paths related to the target timing with the posterior distribution of PVs and evaluate prediction error e'. If it meets e, I finish the exploration; otherwise, I increase the total number by Δn and repeat the whole process until it satisfies the constraint.

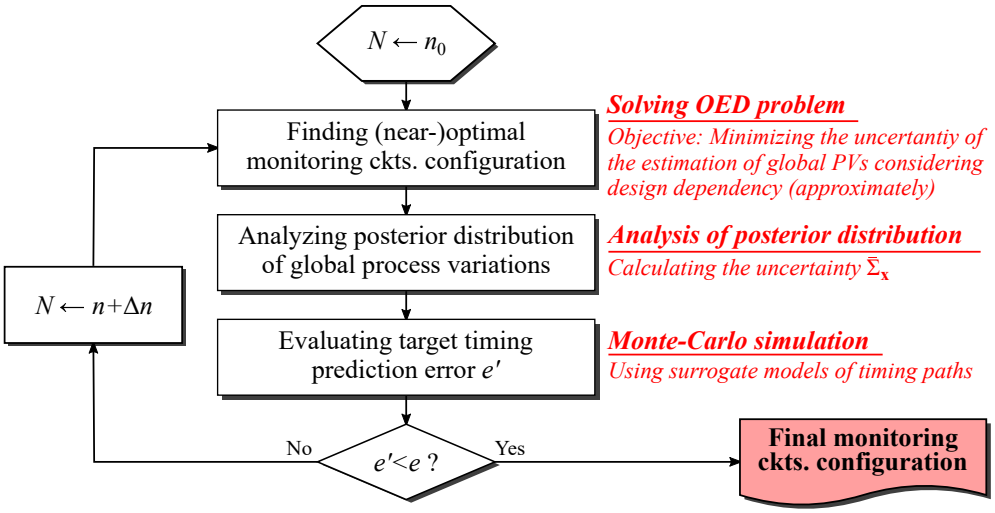


Figure 3.6: Overall optimization flow of the configuration of monitoring circuits in my HPM methodology.

3.3.3 PV2CPT Model Construction

Though most deep learning models only predict their target output for a given input, the information about the confidence of the prediction is necessary as well for many applications. One of the representative researches that addressed the uncertainty in deep learning models is the work of Gal [105], in which the author classifies it into two groups: aleatoric uncertainty (from data noise) and epistemic uncertainty (inherent in a model itself). Later, Kendall and Gal [10] proposed the training method considering them with the neural network model that has two parts of outputs, as shown in Figure 3.7(a). On the other hand, Lakshminarayanan, Pritzel, and Blundell [106] suggested another uncertainty estimation technique in a practical perspective. Like Kendall and Gal, they also split prediction outputs, but they assumed that the outputs follow Gaussian distributions. In addition, they set a negative log-likelihood to their minimization objective, exploited adversarial training for smoothing prediction results, and employed an ensemble model.

In my HPM methodology, pessimism in timing prediction could be excessive if I

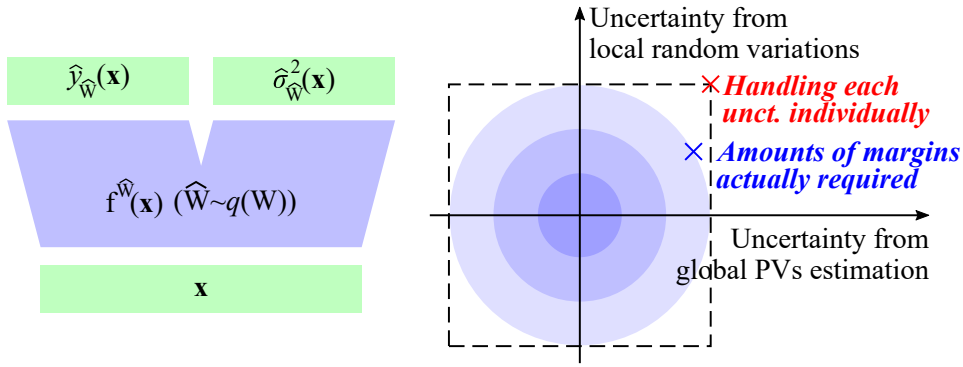


Figure 3.7: (a) A neural network model for evaluating the confidence of target prediction proposed in [10]. The outputs consist of two parts: one for predicting target output $\hat{y}_{\widehat{W}}(\mathbf{x})$ and the other for its uncertainty $\hat{\sigma}_{\widehat{W}}(\mathbf{x})$ given input $\mathbf{x}$. Note that $\widehat{W}$ is random samples from the approximation of the posterior distribution of parameters $q(W)$. (b) Illustration of the inclusion of excessive prediction pessimism when each uncertainty (from global PVs estimation and local random variations themselves) is handled individually.

handle the uncertainty of global PVs estimation and local PVs separately. For example, if I insert margins to cover 3σ points of them individually, the total amount of pessimism included in a final prediction (red X in Figure 3.7(b)) will be much larger than that I require (blue X in Figure 3.7(b)). Thus, in the proposed HPM methodology, I overcome this limitation by applying the concept of uncertainty learning in [105, 10, 106]. Specifically, I employ the ensemble model used in [106] and train it with the input dataset consisting of the pairs of global PVs and corresponding target timings. Note that for considering the estimation uncertainty of global PVs, I use MAP estimation results obtained from an HPM2PV model as the input to the ensemble. Likewise, I intentionally include local PVs for learning them while preparing the dataset through a surrogate model based Monte-Carlo simulation.

3.4 HPM Methodology: Post-Silicon Phase

3.4.1 Transfer Learning in Silicon Characterization Step

Since I constructed HPM2PV and PV2CPT models from PDK based simulation results in a design phase, there can be discrepancies between prediction results and silicon measurements due to various factors. For example, nominal status and variation properties of process parameters in chips can be different from those assumed in PDK by little change of manufacturing condition. Besides, the accuracy of the prediction might decrease through various kinds of models. The difference will likely be more severe when considering voltage and temperature variations and the technical limitation of measurement resolution additionally. In my methodology, to narrow this gap, I exploit the concept of transfer learning for calibrating the prediction model.

In general, machine learning models are trained and used on the assumption of drawing samples from the same feature space and distribution, which is impractical in some applications. The simplest solution is recollecting samples and training the model from scratch despite the expensive cost. Transfer learning is the concept to resolve this issue, which enables engineers to construct a new prediction model efficiently by exploiting the knowledge obtained previously [107, 108]. Yosinski, Clune, Bengio, and Lipson [109] discussed the generality and specificity of the layers in a convolutional neural network model. In the semiconductor domain, Kang [110] proposed the method of exploiting the knowledge from previous test equipment environments to lessen the effort of dataset preparation for a new virtual metrology model. Recently, Lin *et al.* [111] employed the concept of transfer learning to deal with the resist model in lithography simulations.

Similar to [109, 110, 111], I implement my transfer learning flow combining throughout a design phase and post-silicon phase, as shown in Figure 3.8. First, I construct a PV2CPT model using sufficient amounts of data obtained from PDK based simulation results in a design phase. After that, in the following silicon characterization step, I

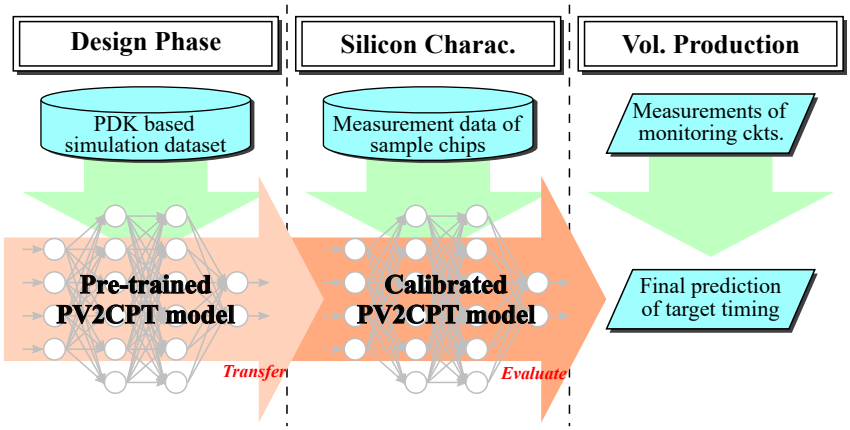


Figure 3.8: Calibration flow of target timing prediction model in my HPM methodology throughout a design phase (left), silicon characterization step (middle), and volume production phase (right).

calibrate the model using the measurements of sample chips. Precisely, I first attempt to fix the parameters in the first layer of the pre-trained PV2CPT model, fine-tune the others in the rest layers and evaluate prediction accuracy with overfitting check. Next, I repeat the same procedures except for fixing the parameters in the first two layers and compare the result with the previous one. In the same way, I fix the parameters in the first three layers, four layers, $\cdots$, and obtain the best PV2CPT model among them at last. I do not tune an HPM2PV model since its basis is a delicate statistical analysis and related to PVs, which usually have a high level of generality. Note that it is possible to conduct some tests such as V_{dd} and f_{max} sweepings in a silicon characterization step, which are almost impractical in a volume production phase in terms of test cost.

3.4.2 Procedures in Volume Production Phase

In a volume production phase, I first calculate the MAP estimation of PVs of a chip using an HPM2PV model constructed in a design phase. From the results, I predict the average of a target timing and its uncertainty using a calibrated PV2CPT model. Then, I predict the target timing pessimistically by adding some timing margin that

corresponds to the prediction yield I aim to achieve to the average at last. For example, if I set the maximum operating frequency f_{max} to my prediction target and aim to achieve prediction yield of 99%, i.e., 99 in 100 chips are safely operated with predicted frequencies, then my final prediction will be the sum of the average and the standard deviation multiplied by $\mathrm{F}^{-1}(0.99)$, where F is the cumulative distribution function of the standard normal distribution. In case of applying AVS in conjunction with my HPM methodology with f_{max}, I increase V_{dd} from the lowest level among the candidates until the operation meets a target frequency f_{target}, i.e., $f_{\text{target}} \geq f_{\text{max}}$, to find the minimum required supply voltage $V_{\text{dd,min}}$.

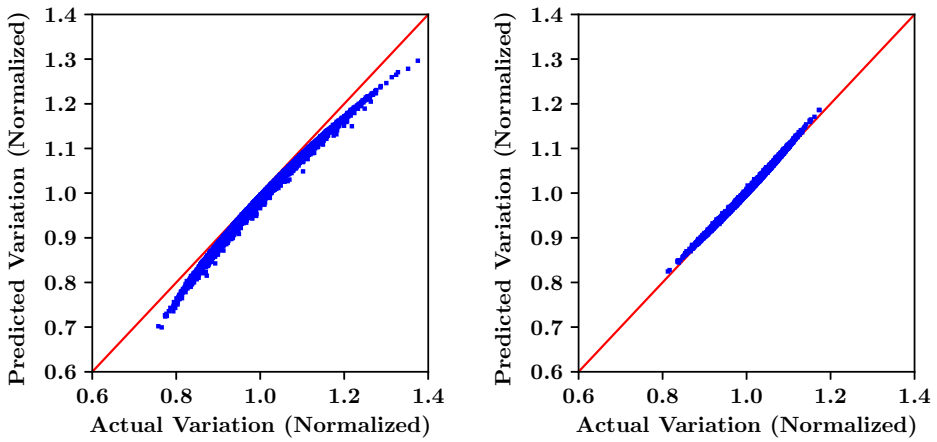
3.5 Experimental Results

3.5.1 Experimental Setup

To validate the efficacy of my HPM methodology in an NTV regime, I used a 28nm industry PDK and DK, characterized at 0.6V of V_{dd}[1] for simulations (from Section. 3.5.2 to Section 3.5.5). I also tested the effectiveness of my prediction model calibration method using test chips fabricated by 10nm process technology (Section 3.5.6). Note that the characterization of PDK/DK and timing closure of the test vehicle proceeded at V_{dd}=0.75V in the super-V_{th} regime instead of the NTV regime for the 10nm process. In addition, as shown in Figure 3.9, I could not observe the nonlinearity of performance variation for the 10nm process in the super-V_{th} regime, which is out of the interest of my HPM methodology. For this reason and confidentiality issue, I only report the experimental results of model calibration in this section.

For the simulation experiments with the 28nm process, I considered 19 FEOL process parameters (e.g., polysilicon gate length) and 20 BEOL process parameters (e.g., metal resistance of M3 layer). I assumed 12 types of ring oscillators for HPM methodologies with no consideration of BEOL PVs, by combining a few kinds of

[1]The typical V_{dd} is 1.0V.



(a) 28nm NTV operation

(b) 10nm super-V_{th} operation

Figure 3.9: Comparison between actual (from surrogate models) and predicted (through HPM methodology) timing variation for (a) 28nm process NTV operation and (b) 10nm process super-V_{th} operation. Blue squares and red lines indicate test results and the ideal prediction (i.e., $y = x$), respectively. Note that I intentionally exclude additional pessimism in prediction results for checking the trend of discrepancies.

cell types and driving strengths (BUF:1∼3, INV:1∼4, DELAY:1∼2, MUX, NAND2, and NOR2). Additionally, I added 20 types of ring oscillators for considering BEOL by inserting snake paths and stacked vias. I will call them FEOL ring oscillators and BEOL ring oscillators, respectively. Table 3.1 lists the benchmark circuits used in my 28nm process experiments. After logic synthesis, placement and routing, and timing closure with 0.6V of V_{dd}, I extracted setup timing critical paths (100 for SPARC and 20 for the others) and generated their surrogate model training datasets through Synopsys PrimeTime and FineSim, respectively.

Likewise, for the 10nm process experiments, I extracted 250 setup timing critical paths per each corner and analyzed their timing sensitivities to FEOL and BEOL PVs through Cadence Tempus and Synopsys FineSim, respectively. Then, I clustered the

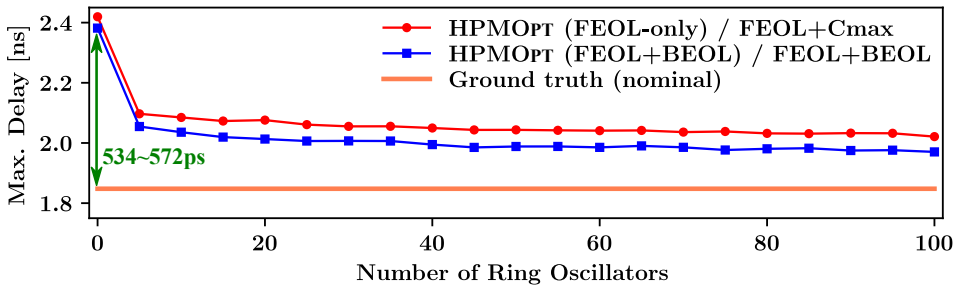
Table 3.1: Benchmark circuits used in my experiments of 28nm process NTV operation.

Design	Description	Freq [MHz]	#Cells
SPARC	Microprocessor core of OpenSPARC T1	282	138,343
aes_cipher	AES cipher (encrypt) block	282	17,760
aes_inv_cipher	AES inverted cipher (decrypt) block	238	23,539
des_perf_opt	Performance-optimized DES block	300	21,087
usb_phy	USB 1.1 PHY	667	557
wb_dma	DMA/Bridge IP core	500	3,823

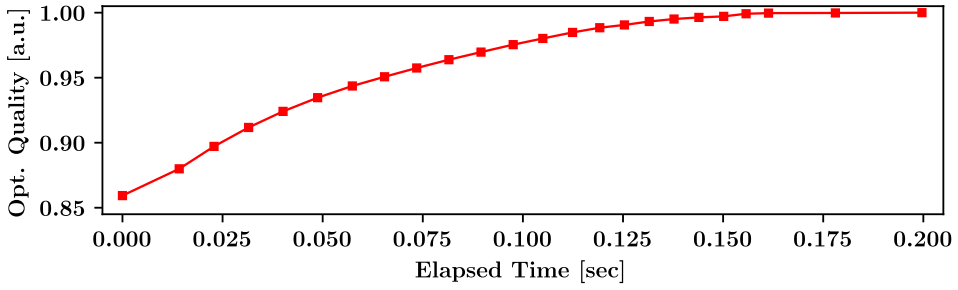
setup timing critical paths based on the sensitivities, as suggested in [47]. After that, I trained surrogate models of 100 timing paths that correspond to the clusters. The total number of sample chips was about 300, and there were 25 instances per each type of ring oscillator over a chip. After the fabrication, I measured the frequencies of the ring oscillators and the lowest supply voltage $V_{\mathrm{dd,min}}$ for each chip at which it can operate correctly at the signoff frequency without any malfunction.

3.5.2 Exploration of Monitoring Circuits Configuration

I first explored the change of prediction pessimism when the optimized monitoring circuits (i.e., ring oscillators in my experiments) configuration was used in my HPM methodology as the total number increases with the step size of 5. I assumed the existence of FEOL and BEOL PVs and did not consider the effect of local random variations in final prediction pessimism. In addition, I supposed the ground truth of the exploration as the maximum datapath delay with no FEOL and BEOL PVs, i.e., nominal condition. The number of surrogate model based Monte-Carlo simulation samples was 10K. Figure 3.10(a) shows the value of 99.87% (3σ) among the results. Note that the label 'HPMOPT (FEOL-only) / FEOL+Cmax' denotes no consideration of BEOL PVs while optimizing the configuration and constructing HPM2PV and PV2CPT models. On the other hand, the label 'HPMOPT (FEOL+BEOL) / FEOL+BEOL' indicates



(a)



(b)

Figure 3.10: (a) Trend of the expected prediction pessimism as the total number of ring oscillator instances increases. (b) Change of the normalized optimization quality as search time elapses. The y-axis denotes the objective value normalized to that of the last result found during the optimization, so the closer the point is to $y = 1$, the more effective the monitoring circuits configuration is.

the inclusion of FEOL and BEOL PVs in the optimization and prediction models construction.

Figure 3.10(a) shows that when I do not exploit HPM methodology (i.e, the number of ring oscillators is 0), the prediction pessimism is about 28.0%∼31.0% of the target maximum delay. However, with 100 ring oscillators and my HPM methodology, it reduces to 9.4% by only considering FEOL PVs and 6.6% by reflecting BEOL PVs additionally. I can also observe that the reduction rate of prediction pessimism decreases gradually as the total number increases. Figure 3.10(b) illustrates the change of the monitoring circuits optimization (HPMOpt (FEOL-only)) quality over elapsed

time when I restricted the total number to 50. From the figure, I observe that my optimizer found the solution of which the optimization gap with the final result is less than 3% after 0.10 seconds, and there was no update of the solution after 0.20 seconds. In other words, by applying the method explained in Section 3.3.2, it is possible to search the (near-)optimal configuration of monitoring circuits in a short time.

Table 3.2 shows the optimized configurations of monitoring circuits (HPMOPT (FEOL-only)) for the benchmark circuits listed in Table 3.1 with the restriction of the use of 50 ring oscillator instances. All the configurations of monitoring circuits seem to be similar on the whole, but there are some differences in part. For example, a few types of ring oscillators (e.g., BUF:3, INV:1, MUX, NOR2) accounted for the majority of the monitoring circuits while others (e.g., BUF:2, INV:2, INV:4, DELAY:1, DELAY:2) were not used for all designs in common. In addition, 8 NOR2 and 18 MUX were used in the result for SPARC, but their numbers changed to 5 and 22, respectively, for wb_dma. In other words, the results demonstrate that my optimization considers the variational behavior of timing paths for different designs (i.e., design dependency) as well as the common properties (e.g., process technology, supply voltage, design size).

3.5.3 Effectiveness of Monitoring Circuits Optimization

For demonstrating the effectiveness of my monitoring circuits optimization, I compared three groups of the configurations with the use of 50 ring oscillator instances: RANDOM (randomly generated), SINGLETYPE (consisted of the same type of ring oscillators), and HPMOPT (my optimization result). Target design was SPARC core, and I considered the existence of FEOL PVs only (i.e., no BEOL PVs and local random variations). The size of training and validation datasets for PV2CPT models were 100K and 20K, respectively. Figure 3.11 and Table 3.3 show statistics on the prediction results with 10K test samples. It can be seen that the average and standard deviation of prediction errors for RANDOM are 4.6%$\sim$25.2% and 4.3%$\sim$24.2% larger than those of HPMOPT, respectively. The gaps are more significant with the configurations be-

Table 3.2: Optimized configurations of ring oscillators for the benchmark circuits listed in Table 3.1.

Design	Runtime [sec]	#BUF:1	#BUF:3	#INV:1	#INV:3	#MUX	#NAND2	#NOR2	#Others	#Total
SPARC	0.20	1	11	7	2	18	3	8	0	50
aes_cipher	0.27	1	10	11	0	21	1	6	0	50
aes_inv_cipher	0.26	2	11	9	0	20	2	6	0	50
des_perf_opt	0.24	1	12	8	0	20	2	7	0	50
usb_phy	0.24	1	12	8	0	20	2	7	0	50
wb_dma	0.24	1	10	11	0	22	1	5	0	50

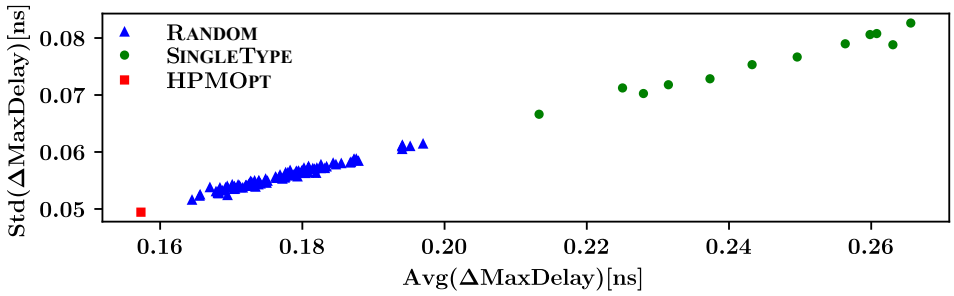


Figure 3.11: Comparison of the average and standard deviation of maximum delay prediction errors for different configurations of ring oscillators.

Table 3.3: Statistics of the results in Figure 3.11. All ΔMaxDelay values are normalized to the results of HPMOPT.

Configuration	ΔMaxDelay [a.u.]		Yield [%]
	Avg.	Std.	
RANDOM	1.046-1.252	1.043-1.242	99.85-99.98
SINGLETYPE	1.356-1.689	1.348-1.671	99.91-99.98
HPMOPT	1.000	1.000	99.90

long to SINGLETYPE, of which the average and standard deviation are 35.6%∼68.8% and 34.8%∼67.1% larger in comparison with the results of HPMOPT, respectively.

3.5.4 Considering BEOL PVs and Uncertainty Learning

Table 3.4 shows the effect of considering BEOL PVs and exploitation of uncertainty learning when there exist BEOL PVs and local random variations as well as FEOL PVs. In Table 3.4, 'FEOL-only' and 'FEOL+BEOL' indicate the consideration of FEOL PVs only and the BEOL PVs additionally, respectively. In addition, '$+3\sigma$ Local' and 'Uncertainty Learning' represent handling uncertainties of local random variations by simply adding some timing margin and integrating it with the estimation uncertainty of global PVs through uncertainty learning, respectively. The other experimental setups (e.g., target design) are all the same as those of Section 3.5.3.

Table 3.4: Comparison of the prediction results when considering BEOL PVs and exploiting uncertainty learning additionally. Note that all values in parentheses are normalized to the results of 'FEOL+BEOL / Uncertainty Learning.'

PVs / Prediction Flow	ΔMax.Delay [ps] ([a.u.])		Yield [%]
	Avg.	Std.	
FEOL-only / $+3\sigma$ Local	246.048 (1.720)	63.362 (1.278)	99.99
FEOL+BEOL / $+3\sigma$ Local	189.921 (1.327)	51.651 (1.042)	100.00
FEOL+BEOL / Uncertainty Learning	143.087 (1.000)	49.563 (1.000)	99.78

The experimental results in Table 3.4 demonstrate that the average and standard deviation of prediction errors can significantly reduce by 22.8% and 18.5%, respectively, through the consideration of BEOL PVs. It also reveals that the conventional timing margin insertion causes excessively pessimistic timing prediction results in comparison with my timing margin control through the uncertainty learning technique. Precisely, the average and standard deviation of prediction errors reduce by 24.7% and 4.0%, respectively, at the expense of little yield drop from 100.00% to 99.78%. However, this drop is not that critical since the target prediction yield was 99.87% (3σ) in my experiments, which is slightly higher than 99.78%.

3.5.5 Comparison among Different Prediction Flows

Table 3.5 summarizes the comparison among a few kinds of target timing prediction flows, including my approach (PROPOSED). STATISTICAL is a method that exploits the statistical model from the measurements of monitoring circuits to target timing prediction, and ML-BASED is a neural network model with no consideration of global PVs. On the other hand, HPM-AVG interpolates the changes of target timing using the average of the measurements of monitoring circuits. SLOW-SLOW is the signoff results with SS corner and Cmax corner. Although SLOW-SLOW is not an HPM methodology, I include its results to validate the effect of HPM methodologies compared to the conventional signoff approach. All the other experimental setups and procedures except

Table 3.5: Comparison among different target timing prediction flows (i.e., STATISTICAL, ML-BASED, and PROPOSED) and conventional signoff results (SLOW-SLOW). All values in parentheses are normalized to the results of PROPOSED.

Prediction Flow	ΔMax.Delay [ps] ([a.u.])		Yield [%]
	Avg.	Std.	
SLOW-SLOW (no offset)	647.821 (4.527)	150.798 (3.043)	99.98
STATISTICAL (no offset)	117.450 (0.821)	48.626 (0.981)	99.10
ML-BASED (no offset)	123.917 (0.866)	54.179 (1.093)	98.79
HPM-AVG (no offset)	213.420 (1.492)	71.367 (1.440)	99.70
PROPOSED (no offset)	143.087 (1.000)	49.563 (1.000)	99.78
SLOW-SLOW (with offset)	517.576 (3.427)	150.798 (3.043)	99.87
STATISTICAL (with offset)	155.576 (1.039)	48.626 (0.981)	99.87
ML-BASED (with offset)	170.379 (1.129)	54.179 (1.093)	99.87
HPM-AVG (with offset)	230.351 (1.526)	71.367 (1.440)	99.87
PROPOSED (with offset)	150.908 (1.000)	49.563 (1.000)	99.87

for target timing prediction are the same as those in Section 3.5.4. Note that 'no offset' and 'with offset' mean the prediction results with no post-processing and after addition of some margin to set yields to the target (99.87% (3σ) in my experiments) intentionally, respectively.

From the results without additional offset, I observe that STATISTICAL shows the smallest prediction errors. However, its prediction yield is 99.10%, which is much lower than the target yield 99.87% and the prediction yield of PROPOSED. When I set all the prediction yields to the target 99.87% for a fair comparison, I can see that PROPOSED outperforms the other prediction flows in terms of the average prediction error. Specifically, it is smaller than those of STATISTICAL and ML-BASED by 3.0% and 11.4%, respectively. In comparison with the conventional signoff result (SLOW-SLOW), PROPOSED recovers 77.9% of pessimism for the maximum delay on average. The results of HPM-AVG show the reason why elaborated HPM methodology is essential for reducing timing prediction pessimism.

Figure 3.12(a) compares the controllability of timing prediction margin for STA-

TISTICAL and PROPOSED. STATISTICAL tends to predict the maximum delays optimistically throughout all target yields (from 50.00% to 99.87%), while PROPOSED maintains prediction yields almost similar to the ideal case. The main reason for a large amount of optimism in STATISTICAL is the nonlinearity of performance variation shown in Figure 3.9(a). Figure 3.12(b) shows the change of prediction yield with no offset as dataset size increases, from which I can see that PROPOSED converges to the target yield about 12.5× faster than ML-BASED. Note that I set the size of a validation dataset to 1/5× of that of a training dataset while remaining all the others as before. In short, my approach exhibits more accurate control of timing pessimism considering target prediction yield than other prediction flows even with small datasets.

3.5.6 Effectiveness of Prediction Model Calibration

Figure 3.13(a) illustrates prediction results in conjunction with AVS before and after calibration of the prediction model in the silicon characterization step for the 10nm process technology. The numbers of training and validation samples for fine-tuning were 15K and 5K, respectively, and I tested on 8.9K samples[2]. Prediction results with no calibration (blue circles) cannot follow the trend of $V_{\mathrm{dd,min}}$ change well due to the reasons I listed in Section 3.4.1. However, the gap is much narrower after the fine-tuning of the pre-trained model (red squares). Statistics on differences between actual and predicted $V_{\mathrm{dd,min}}$ are shown in the first two rows in Table 3.6, in which the average error decreases by 65.81% while maintaining the same level of prediction yield.

In real silicon manufacturing environments, it is impractical to gather such a large amount of data. Hence I validated the effectiveness of my calibration when the numbers of training and validation samples are only 250 and 50, respectively. As shown in Figure 3.13(b), the result is almost the same in comparison with that in Figure 3.13(a) except for the slight drop of prediction yield from 100.00% to 99.46%, which is a little

[2]I inflated the number of samples by combining the measurements of the same type of ring oscillators on a chip.

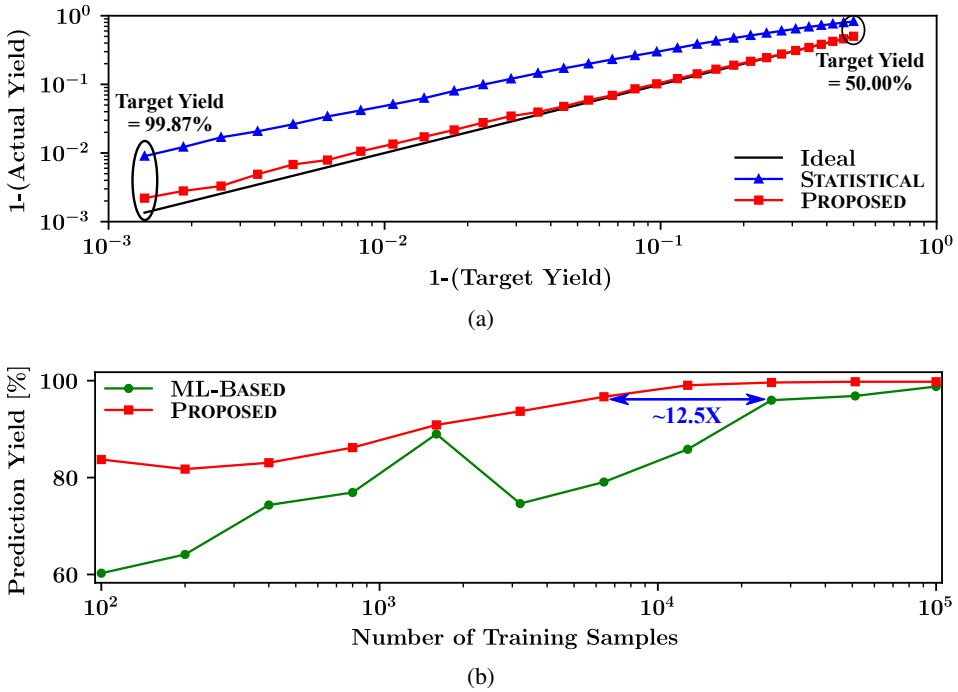
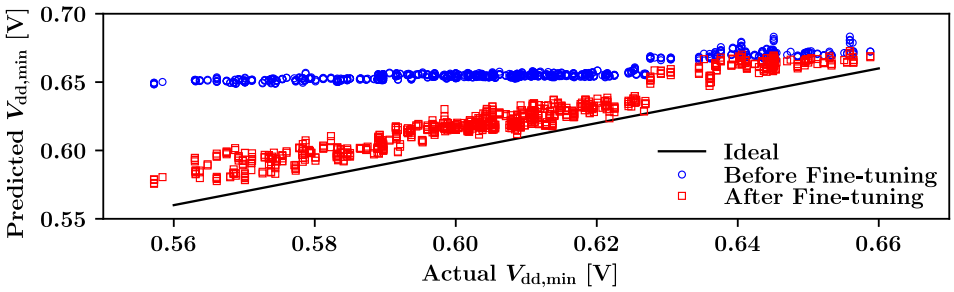


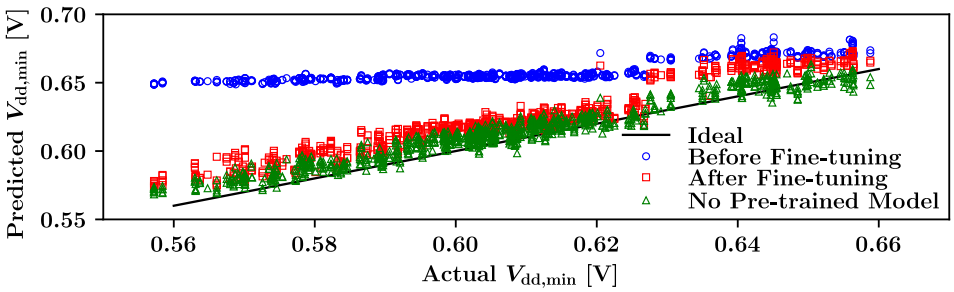
Figure 3.12: Comparison between different target timing prediction flows. (a) Controllability of timing pessimism as target prediction yield varies. The x- and y-axis denote the expected fail ratio (i.e., 1-(target prediction yield)) and the actual fail ratio (i.e., 1-(actual prediction yield)), respectively, and the black line represents the ideal case that expectation yields perfectly match with prediction results. (b) The convergence of prediction yield as the number of training samples increases.

bit smaller than the target yield 99.87% (3σ). I also trained a neural network model with the same dataset from scratch, of which the evaluation result is in the last row in Table 3.6 ('Small / No Pre-trained Model'). The average prediction error and standard deviation are smaller than those of the fine-tuned model, but the prediction yield is only 86.57%, which is unacceptable in the industry.

Meanwhile, when the number of training and validation samples are 15K and 5K, respectively, dynamic power consumption reduces 21.4% and 28.2% on average



(a) Number of training/validation samples: 15K/5K



(b) Number of training/validation samples: 250/50

Figure 3.13: Comparison of $V_{dd,min}$ prediction results before and after fine-tuning using data measured in silicon characterization step. The x- and y-axis denote the actual and predicted $V_{dd,min}$, and the black line represents the ideal prediction.

through the pre-trained model before and after fine-tuning in silicon characterization step, in comparison with the typical V_{dd} operation of every chip. Note that I set the resolution of V_{dd} change to 20mV. Even for the case with 250 training and 50 validation samples, the amount of power-saving is almost the same. Precisely, decreases of dynamic power consumption before and after the calibration are 21.4% and 28.8% on average, respectively.

3.6 Summary

In this chapter, I proposed a holistic HPM methodology from design to post-silicon phase for handling wide and nonlinear performance variation in an NTV regime and

Table 3.6: Statistics on the results in Figure 3.13. All values in parentheses are normalized to the results of 'Small / After Fine-tuning.'

Dataset Size / Prediction	$\Delta V_{\text{dd,min}}$ [mV] ([a.u.])		Yield [%]
	Avg.	Std.	
Large / Before Fine-tuning	49.407 (3.38)	18.222 (2.83)	100.00
Large / After Fine-tuning	16.894 (1.16)	5.960 (0.92)	100.00
Small / Before Fine-tuning	49.427 (3.38)	18.290 (2.84)	100.00
Small / After Fine-tuning	14.620 (1.00)	6.450 (1.00)	99.46
Small / No Pre-trained Model	5.995 (0.38)	5.465 (0.85)	86.57

advanced process to reduce timing prediction pessimism. Precisely, (1) I formulated the problem of finding an efficient configuration of monitoring circuits into the optimal experiment design problem and (2) proposed a new timing prediction flow by combining statistical estimation of FEOL and BEOL PVs and a neural network based timing inference model. In addition, (3) I introduced uncertainty learning for accurate control of timing margin and transfer learning for the calibration of the prediction model. Furthermore, (4) I replaced time-consuming SPICE simulations in my methodology with efficient but accurate surrogate models. Through experiments with a 28nm industry PDK and DK characterized at 0.6V, I reduced the average prediction pessimism of maximum delay by 77.9% over the conventional signoff results while respecting the target prediction yield. I also demonstrated that my holistic approach, in conjunction with AVS, could save dynamic power consumption by 28.2%$\sim$28.8% on average for test chips fabricated using a 10nm process.

Chapter 4

LIGHTENING ASYNCHRONOUS PIPELINE CONTROLLER

4.1 Preliminaries and State-of-the-Art Work

4.1.1 Bundled-data vs. Dual-rail Asynchronous Circuits

Bundled-data encoding is a coding style that transmits each data bit using exactly one signal wire, for which a handshaking mechanism through request and acknowledgement signal wires between two components should be installed, as shown in Figure 4.1. A bundled-data asynchronous circuit consists of a datapath, which is the same structure as a synchronous circuit, and a controller. It is particularly attractive since it has relatively less area overhead over its dual-rail counterpart. However, since the delays of handshaking signals (*req*, *ack*) control all the timings of datapath operation, sufficient timing margins should be allotted to endure variation. Contrary to bundled-data encoding, dual-rail encoding is a technique that entails a data value and its validity simultaneously by using two signal wires for every data bit. Thus, it is more robust to timing variation with no timing margins. Though it requires no external handshaking logic, it incurs a substantial area overhead because of doubling signal wires and detection logic for checking the completion of logic operations.

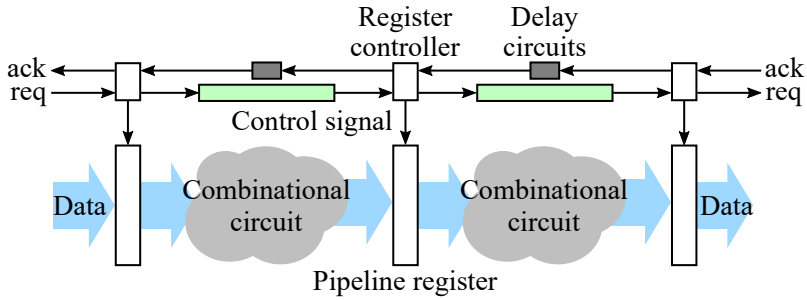
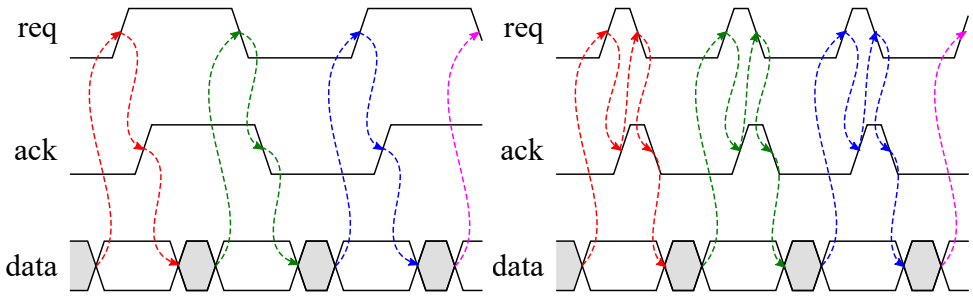


Figure 4.1: Structure of a bundled-data asynchronous pipeline circuit. A delay circuit, e.g., a delay buffer chain, should be inserted on each pipeline stage (the long green and short gray bars) to build up the setup and hold timing paths.

4.1.2 Two-phase vs. Four-phase Bundled-data Protocol

Figure 4.2(a) shows how a two-phase bundled-data protocol operates. A transaction starts with issuing data and making a transition on a request signal by a sender. When the receiver accepts this signal, it starts to read the transaction data and finishes the transaction by making a transition on its acknowledgement signal. On the other hand, a four-phase bundled-data protocol operates as shown in Figure 4.2(b): first, a sender issues data and sets the request signal to 1. Then, the receiver detects this signal and begins to read data while setting the acknowledgement signal to 1. The sender accepts this acknowledgement signal, initializes the request signal to 0, and at last, the receiver follows it by setting the acknowledgement signal to 0, completing the transaction.



(a) Two-phase (b) Four-phase

Figure 4.2: Two types of bundled-data handshaking protocols.

4.1.3 Conventional State-of-the-Art Pipeline Controller Template

Figure 4.3 shows a part of the state-of-the-art asynchronous pipeline controller template in [3] to be installed on each pipeline stage. The structure consists of two XOR gates, one NOR gate, and a resettable transparent latch (pink box), all of which are connected to support the communication protocol on that pipeline stage. Procedures of the transactions are as follows: let me assume that all request and acknowledgement signals are 0 initially. Then the upper XOR and the lower XOR are 1 and 0, respectively, and thus, the NOR is 0, which makes the latch close. As $\text{req}^{(i-1)}$ becomes 1 at the latch input, the upper XOR goes to 0, causing NOR to 1, which makes the latch transparent. After that, the lower XOR becomes 1, causing NOR to 0, which makes the latch close again. This short time interval (i.e., the sum of the NOR and the lower XOR delays) during which the latch is transparent, thus reducing glitches, is the most significant advantage of this controller[1]. After a relatively long time through the delay circuit, the request signal req^i of logic value 1 goes to the controller on pipeline stage $i+1$ and comes back as the acknowledgement signal $\text{ack}^{(i+1)}$, which completes the transaction on pipeline stage i. Then it initiates the next event on pipeline stage i with the opposite polarities, i.e., the request and acknowledgement signals of logic value 1.

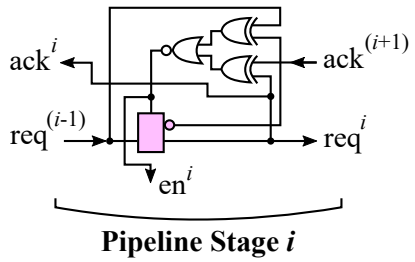


Figure 4.3: Asynchronous pipeline controller template proposed in [3]. (For simplicity, I omit a delay circuit on the request signal line right after the latch in this structure.)

[1]Note that my proposed pipeline template never enlarges this time interval.

4.2 Delay Path Sharing for Lightening Pipeline Controller Template

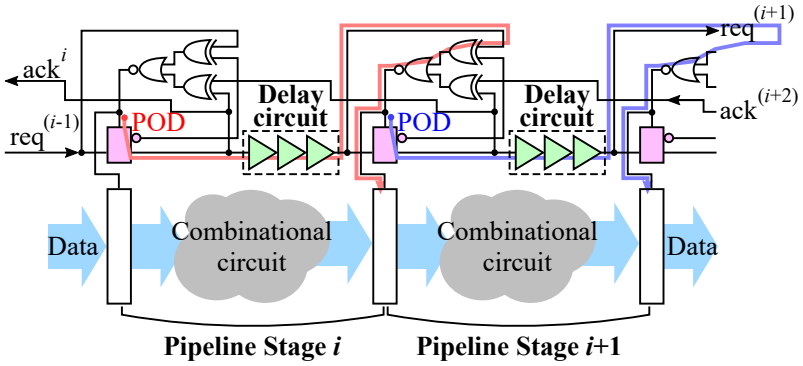
4.2.1 Synthesizing Sharable Delay Paths

Figure 4.4(a) shows a section of an asynchronous pipeline circuit using the conventional controller template in [3], in which the red and blue paths indicate the setup timing paths for pipeline stages i and $i + 1$, respectively. Note that the setup timing path on each pipeline stage should be long enough so that its delay should exceed the timing critical path delay of the combinational circuit on that pipeline stage by inserting a delay circuit such as a delay buffer chain (green triangles in Figure 4.4(a)). Figure 4.4(a) shows that the two setup timing paths in pipeline stages i and $i + 1$ are physically disjoint, which means the total number of delay buffers is the sum of the numbers of delay buffers in those pipeline stages. On the other hand, Figure 4.4(b) shows my pipeline controller template with *sharable delay paths* and its setup timing paths, exhibiting two distinct features:

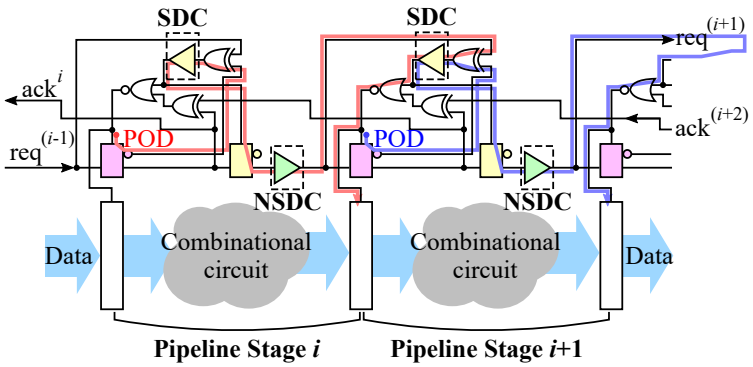
1. The red setup timing path is passing through both delay circuits in pipeline stages i and $i + 1$, while the blue setup timing path is passing through both delay circuits in pipeline stages $i+1$ and $i+2$. Thus, the two setup timing paths partially overlap.

2. Delay circuits, marked as yellow triangles in Figure 4.4(b), are inserted into a template spot between the NOR and the upper XOR. I call such delay circuits *sharing delay circuits (SDCs)* and the rest of delay circuits (i.e., green triangles) *non-sharing delay circuits (NSDCs)*.

Section 4.2.2 will show that although the new setup timing paths are physically conflicting by *Feature 1*, the logical behavior of all the setup timing paths will fulfill the original mission. *Feature 2* then provides an opportunity to reduce the total number of delay buffers by allocating as many of them as possible in SDCs while satisfying all



(a) Using the conventional controller template in [3]



(b) Using my pipeline controller template with sharable delay paths

Figure 4.4: Asynchronous pipeline circuits (a) using the controller template in [3] and (b) using my pipeline controller template with sharable delay paths. The setup timing paths are highlighted in red and blue colors, and the newly added logic cells are marked with yellow color.

necessary timing constraints. I will explain the timing constraints for correct operations on asynchronous pipeline circuits and the detailed formulation of minimizing the total number of delay buffers in Section 4.2.3 and Section 4.2.4, respectively.

It should be noted that some delay circuits are also required at the location of the gray rectangles in Figure 4.1 to build up the hold timing paths for guaranteeing reliable data sampling. However, since local enable networks in an asynchronous pipeline circuit are balanced generally, the number of delay buffers in the delay circuits is small in comparison with that of setup timing paths. Hence I focus on minimizing the number of delay buffers on setup timing paths in this chapter.

4.2.2 Validating Logical Correctness for Sharable Delay Paths

Figure 4.5 shows the circuit structure of my controller template, on which two setup timing paths share the upper XOR and SDC. I also allocate one latch (yellow rectangle) right before feeding NSDC in each pipeline stage. The role of this latch is to prevent the request signal req^i starting from the ordinary controller latch (pink rectangle) from passing through NSDC until the latch enable signal $\text{en}^{i\rightarrow(i+1)}$ coming through the upper XOR and SDC is on. Consequently, the delay of the setup timing path can be manipulated by controlling the numbers of delay buffers in SDC and NSDC. Note that en^i changes to 0 when either of the outputs of two XORs becomes 1. In other words, regardless of the delay of SDC, it is achievable to reduce the glitch power consumption through data transmission like the controller template in [3] since the propagation delay of the path coming through the pink latch and the lower XOR is short enough.

Figure 4.6 shows a part of the waveforms produced by SPICE simulation. The state change on $\text{req}^{(i-1)}$ (crimson wave) enables signal en^i to make its latch (pink box in Figure 4.5) transparent (green wave), which sets $\text{XOR}^i_{\text{upper}}$ back to 1 (pink wave). Then signal $\text{en}^{i\rightarrow(i+1)}$ is enabled to make its latch (yellow box) transparent (purple wave) after a certain period, which is exactly the delay of SDC (yellow triangles). As

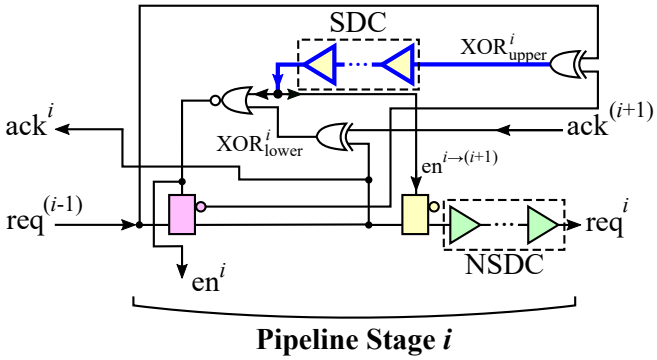


Figure 4.5: The structure of my controller template on a pipeline stage. It is composed of a sharing delay circuit (SDC), a non-sharing delay circuit (NSDC), and a few subsidiary control logic components.

$\text{req}^{(i-1)}$ passes through NSDC (green triangles), it becomes req^{i} (red wave). Figure 4.7 describes all feasible scenarios of the logic behavior of my controller template. Two branches from each of logic states A and B take into account the race between the request (from left) and the acknowledgement (from right) signals. The behavior corresponding to the black and blue dashed boxes reveals that my setup timing path is working correctly for sending 1-state and 0-state of request signals, respectively.

4.2.3 Reformulating Timing Constraints of Controller Template

Timing correctness of a bundled-data asynchronous circuit is composed of two groups of constraints: template-level constraints and protocol-level constraints [112]. The former assures the quasi-delay insensitive assumptions of a controller template and the latter ensures successful data sampling and transmission. In the following, I formulate the two groups of timing constraints. Table 4.1 defines the list of notations used in the formulation.

• **Template-level constraints**: Like the controller template in [3], my controller template with SDCs and NSDCs should also be hazard-free, for which it has to satisfy the following two constraints.

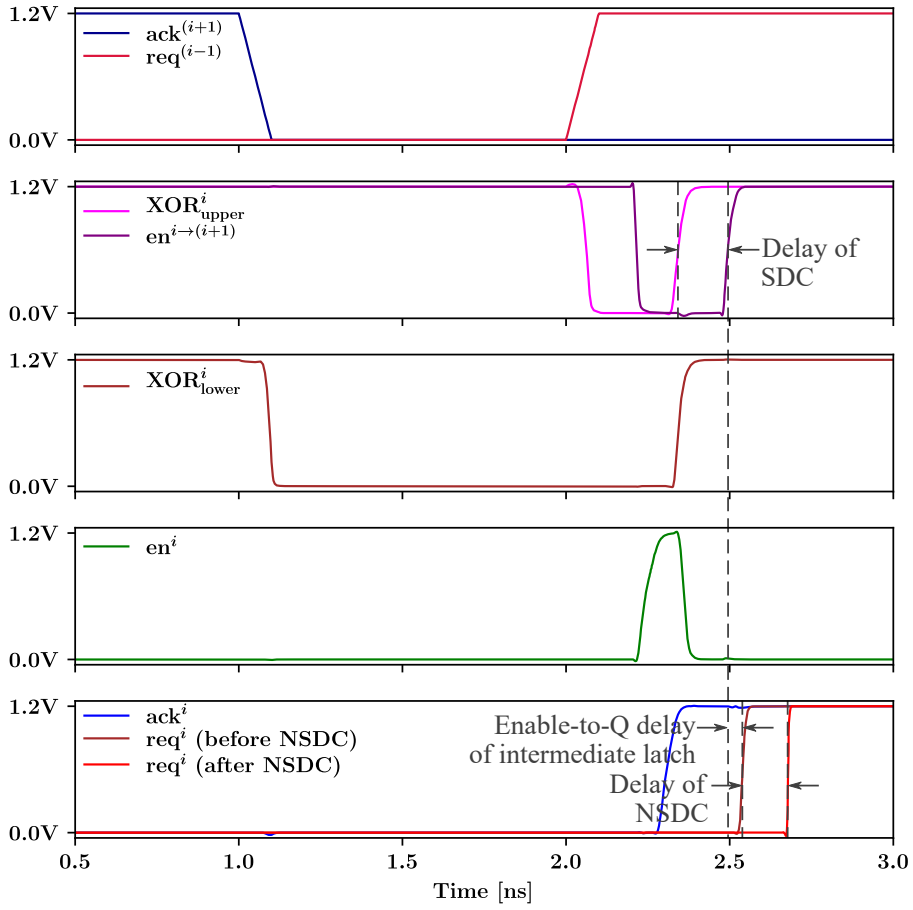


Figure 4.6: Simulation waveforms of my controller template in Figure 4.5.

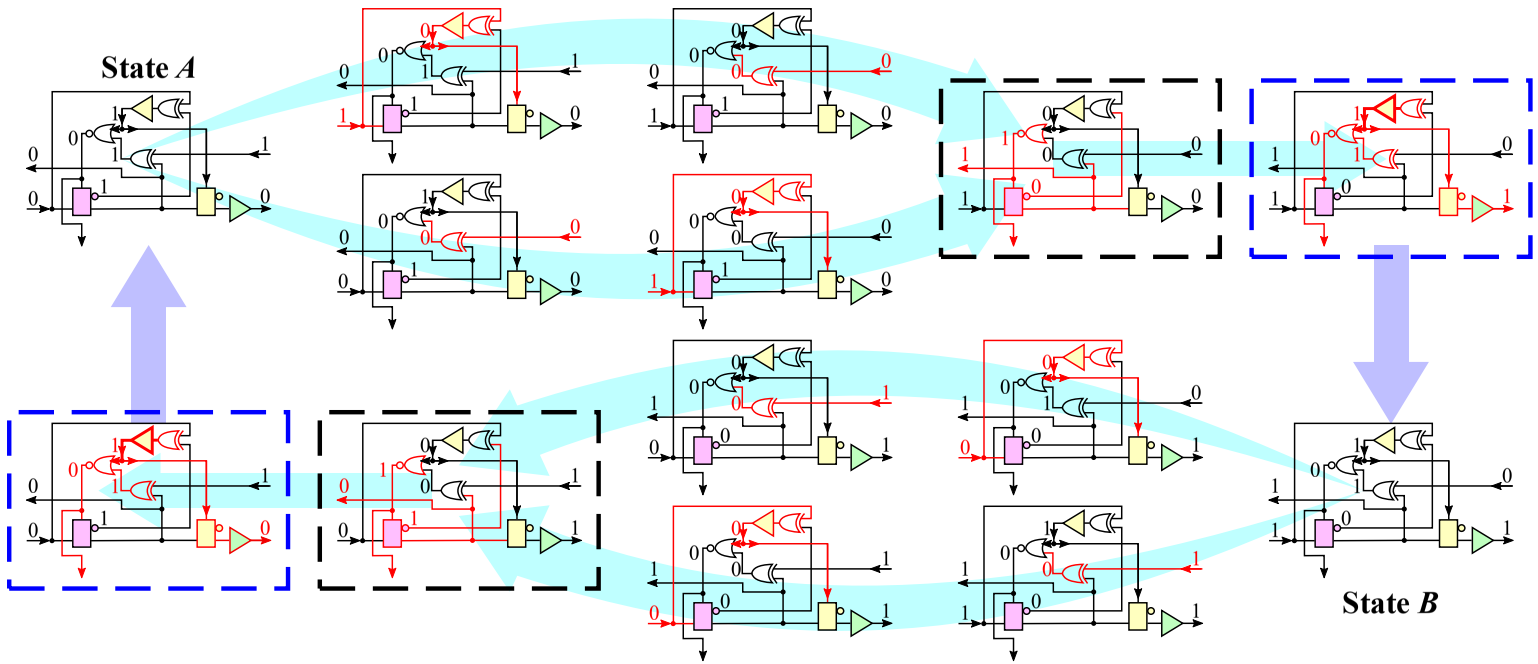


Figure 4.7: Logic behavior of my controller template in Figure 4.5. The parts in red color indicate the change of logic values.

Table 4.1: Definition of notations used in timing constraints.

Terms	Definition
D_R^i	Propagation latency of the request signal on stage i
D_A^i	Propagation latency of the acknowledgement signal on stage i
T_R^i	Request-to-enable delay at the controller on stage i
T_A^i	Acknowledgement-to-enable delay at the controller on stage i
D_L^i	Propagation latency of the enable signal to the sink on stage i
T_C^i	Enable-to-Q delay of the pipeline register on stage i
D_P^i	Delay of the datapath on stage i
T_s^i	Setup time for the pipeline register on stage i
T_h^i	Hold time for the pipeline register on stage i
d_{SDC}^i	Delay corresponding to SDC on stage i
d_{NSDC}^i	Delay corresponding to NSDC on stage i
$d_{\text{latch}}^{i\rightarrow(i+1)}$	Enable-to-Q delay of the newly inserted latch on stage i
$Latency^i$	Latency of stage i in the conventional implementation
$Cycle^i$	Cycle time of stage i in the conventional implementation

Constraint 1: Once the latch on each pipeline stage has begun to accept the request signal from the previous stage, the acknowledgement signal to this pipeline stage should not change its state until the template on that pipeline stage becomes a suspended (steady) state.

I can express this constraint as

$$\begin{aligned}&\max\left\{d_{\text{XOR}}^i + d_{\text{NOR}}^i, d_{\text{XOR}}^i + d_{\text{SDC}}^i + d_{\text{NOR}}^i\right\}\\&\leq d_{\text{XOR}}^i + d_{\text{SDC}}^i + d_{\text{latch}}^{i\rightarrow(i+1)} + d_{\text{NSDC}}^i + d_{\text{XOR}}^{(i+1)} + d_{\text{SDC}}^{(i+1)} + d_{\text{NOR}}^{(i+1)} + d_{\text{latch}}^{(i+1)}.\end{aligned} \tag{4.1}$$

If the delays of the gates of the same type are all identical, I can simplify the inequality in Equation (4.1) as

$$0 \leq d_{\text{XOR}} + 2 \times d_{\text{latch}} + d_{\text{NSDC}}^i + d_{\text{SDC}}^{(i+1)}.$$

Constraint 2: Once the latch on each pipeline stage has begun to accept the request signal from the previous stage, the request signal should not change its state until the

template on that pipeline stage becomes a suspended (steady) state.

I can express this constraint as

$$\begin{aligned}&\min\left\{d_{\text{XOR}}^{(i+1)}+d_{\text{NOR}}^{(i+1)}, d_{\text{XOR}}^{(i+1)}+d_{\text{SDC}}^{(i+1)}+d_{\text{NOR}}^{(i+1)}\right\}\\&\leq d_{\text{XOR}}^{i}+d_{\text{NOR}}^{i}+d_{\text{latch}}^{i}+d_{\text{XOR}}^{i}+d_{\text{SDC}}^{i}+d_{\text{latch}}^{i\rightarrow(i+1)}+d_{\text{NSDC}}^{i}.\end{aligned} \tag{4.2}$$

If the delays of the gates of the same type are all identical, I can simplify the inequality in Equation (4.2) as

$$0 \leq d_{\text{XOR}}+2\times d_{\text{latch}}+d_{\text{SDC}}^{i}+d_{\text{NSDC}}^{i}.$$

Note that it should be guaranteed that there is no hazard on the nets shared by the setup timing paths, which trivially holds when *Constraint 2* is satisfied. In addition, the enable signal to the new latch I inserted should arrive no earlier than the data signal, which is trivial as well.

• **Protocol-level constraints**: For a bundled-data asynchronous pipeline circuit depicted in Figure 4.8, I can express its protocol-level timing constraints as

$$D_R^i+T_R^{(i+1)}+D_L^{(i+1)} \geq D_L^i+T_C^i+D_P^i+T_s^{(i+1)}, \tag{4.3a}$$

$$D_A^i+T_A^i+D_L^i+T_C^i+D_P^i \geq D_L^{(i+1)}+T_h^{(i+1)}. \tag{4.3b}$$

The two inequalities Equations (4.3a) and (4.3b) indicate the setup and hold timing constraints, respectively, for reliable data transmission at the pipeline registers. In the following, I will reformulate the timing constraints based on the two inequalities Equations (4.3a) and (4.3b), which should be satisfied for every asynchronous pipeline circuit that employs my controller template.

(1) The point-of-divergence (POD) of setup timing path on each pipeline stage is the location (red and blue dots in Figure 4.4(b)) at which the enable signal feeds that pipeline stage latch. The launch path starts from the POD and terminates at the data pin to the next pipeline stage registers of an asynchronous circuit, passing through the local enable buffers driving the enable signals to the pipeline registers. On the

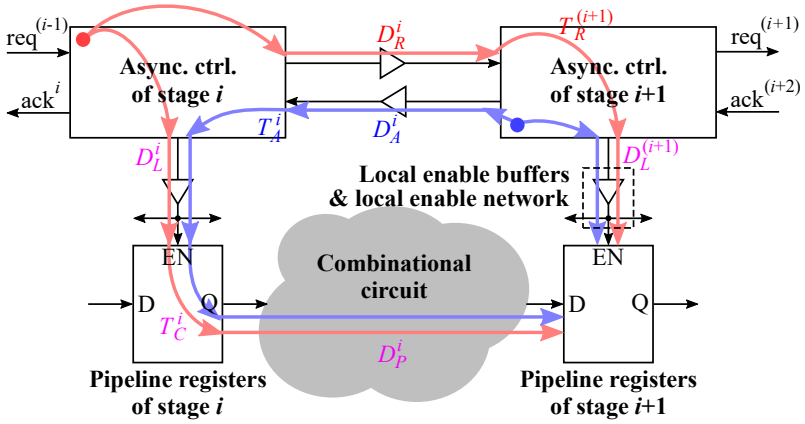


Figure 4.8: The view of timing paths on a bundled-data asynchronous circuit. The red and blue lines denote the launch and capture paths of the setup and hold timing paths between pipeline stages i and $i+1$, respectively.

other hand, the capture path passes through the request signal line and the local enable buffers, and finally terminates at the enable pins to the pipeline registers. Thus, it is possible to rewrite the setup timing constraint in Equation (4.3a) as

$$d^i_{\text{latch}} + d^i_{\text{XOR}} + d^i_{\text{SDC}} + d^{i\rightarrow(i+1)}_{\text{latch}} + d^i_{\text{NSDC}} + d^{(i+1)}_{\text{XOR}} + d^{(i+1)}_{\text{SDC}} + d^{(i+1)}_{\text{NOR}} + D^{(i+1)}_L \geq D^i_L + T^i_C + D^i_P + T^{(i+1)}_s. \tag{4.4}$$

(2) Likewise, the POD of hold timing path on each pipeline stage is the location where an enable feeds the next pipeline stage latch. The launch path starts from the POD, passes through the acknowledgement signal line, the local enable buffers, and the datapath in the current pipeline stage, and terminates at the data pin to the pipeline registers on the next pipeline stage at last. On the other hand, the capture path is quite short, which starts from the same POD, goes through the local enable buffers, and terminates at the enable pin to the pipeline registers on the next pipeline stage. Hence, I can reformulate the hold timing constraint in Equation (4.3b) as

$$d^{(i+1)}_{\text{latch}} + d^i_{\text{XOR}} + d^i_{\text{NOR}} + D^i_L + T^i_C + D^i_P \geq D^{(i+1)}_L + T^{(i+1)}_h. \tag{4.5}$$

4.2.4 Minimally Allocating Delay Buffers

Unlike the conventional asynchronous pipeline controller template, mine flexibly distributes delay buffers to multiple locations of every setup timing path while sharing some delay buffers among the setup timing paths. Consequently, it is possible to minimize the total number of delay buffers in an asynchronous pipeline controller while satisfying the template-level and the protocol-level timing constraints reformulated in Section 4.2.3. Let $\mathcal{S}$=$\{0, 1, 2, \cdots\}$ denote the set of all pipeline stage indexes. If I implement all delay circuits in an asynchronous pipeline controller using delay buffer chains, the propagation delay of the delay circuit will be linearly proportional to the number of delay buffers in it. As a result, I can formulate the equivalent problem of minimizing the total number of delay buffers into a linear programming (LP) as

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in \mathcal{S}} \left(d^i_{\text{SDC}} + d^i_{\text{NSDC}} \right) \\
\text{subject to} \quad & \text{Equations } (4.1), (4.2), (4.4), (4.5), && \forall i \in \mathcal{S}, \\
& d^i_{\text{latch}} + d^i_{\text{XOR}} + d^i_{\text{SDC}} + d^{i \rightarrow (i+1)}_{\text{latch}} \\
& \quad + d^i_{\text{NSDC}} + d^{(i+1)}_{\text{XOR}} + d^{(i+1)}_{\text{SDC}} + d^{(i+1)}_{\text{NOR}} \\
& \quad \leq Latency^i + \epsilon, && \forall i \in \mathcal{S}, \\
& d^i_{\text{latch}} + d^i_{\text{XOR}} + d^i_{\text{SDC}} + d^{i \rightarrow (i+1)}_{\text{latch}} \\
& \quad + d^i_{\text{NSDC}} + d^{(i+1)}_{\text{XOR}} + d^{(i+1)}_{\text{SDC}} + d^{(i+1)}_{\text{NOR}} \\
& \quad + d^{(i+1)}_{\text{latch}} + d^i_{\text{XOR}} + d^i_{\text{NOR}} \\
& \quad \leq Cycle^i + \epsilon, && \forall i \in \mathcal{S}.
\end{aligned}
\tag{4.6}
$$

Note that ϵ is a small value (e.g., 10ps) to avoid the infeasibility of the formulation caused by the discrepancy in gate delay models. The first four ensure the satisfaction of the template-level and the protocol-level timing constraints, and the last two inequalities prevent the performance degradation caused by the inappropriate distribution of delay buffers. For example, when one additional delay buffer is required to meet the setup timing constraint on pipeline stage i, I can distribute it to either d^i_{SDC}, d^i_{NSDC},

or $d_{\mathrm{SDC}}^{(i+1)}$. If all the timing constraints of pipeline stages $i-1$ and $i+1$ have already been satisfied, the allocation of the delay buffer to d_{SDC}^{i} or $d_{\mathrm{SDC}}^{(i+1)}$ will make the cycle time or latency of those pipeline stages unnecessarily longer and finally deteriorate the overall circuit performance. However, with the consideration of the last two inequalities, an LP optimizer is able to select d_{NSDC}^{i} for the delay buffer distribution in such a situation, thereby avoiding the unnecessary degradation of the circuit performance.

4.3 In-depth Pipeline Controller Template Synthesis with Delay Path Reusing

4.3.1 Synthesizing Delay Path Units

In Section 4.2, I converted the asynchronous pipeline controller template in [3] to mine with sharable delay paths for minimizing the total number of delay buffers. In the same way, for reusing the delay buffers, I can apply this conversion technique to any delay buffer chains in SDCs and NSDCs in the controller with slight modification recursively as far as it saves the area. I call those delay circuits produced by the applications of the technique to delay buffer chains *Delay Path Units* (DPUs).

Figure 4.9 illustrates my approach of reducing the number of delay buffers in the controller with the concept of delay path reusing. Figure 4.10 shows three DPU types called DPU-1, DPU-2, and DPU-3, which are the delay circuit structures produced by applying the conversion technique once, twice, and three times recursively on delay buffer chains, respectively. For example, I can obtain DPU-2 by inserting DPU-1 instead of the delay buffer chain in between the upper XOR and NOR (labeled as layer-1 delay buffers) in DPU-1. Similarly, it is possible to make DPU-2 by replacing the delay buffer chain located in between the uppermost XOR and NOR (labeled as layer-2 delay buffers) in DPU-2 with DPU-1. In this way, I can synthesize any arbitrary DPUs. Note that I use DPU-0 to denote a simple delay buffer chain.

The red path in Figure 4.10(b) indicates the input signal flow in DPU-1, which

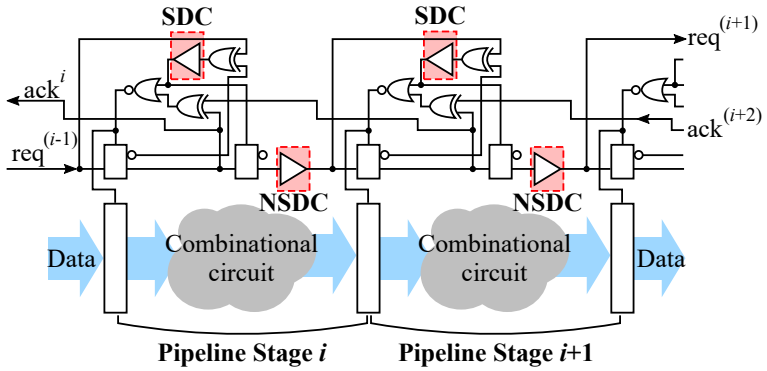
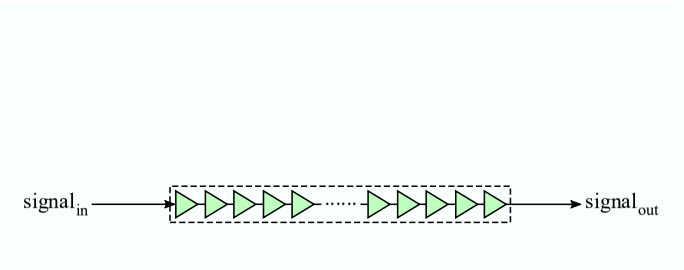


Figure 4.9: Proposed delay circuit implementation approach. I replace SDCs and NSDCs (red dashed boxes) in the controller with new delay circuits called delay path units (DPUs) through the concept of delay path reusing.

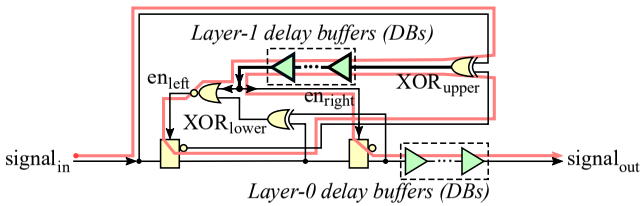
passes the layer-1 delay buffers twice. Likewise, for DPU-2 and DPU-3, the timing paths of input signal propagation pass through the top-layer delay buffers up to 2^2 times and 2^3 times, respectively. By generalizing this, I can state that the timing path of the input signal propagation goes through the delay buffers in DPU-k up to 2^k times. Thus, in comparison with a simple delay buffer chain, the number of delay buffers required in the corresponding DPU-k is theoretically reducible up to $1/2^k$ of that of the simple delay chain.

4.3.2 Validating Logical Correctness of Delay Path Units

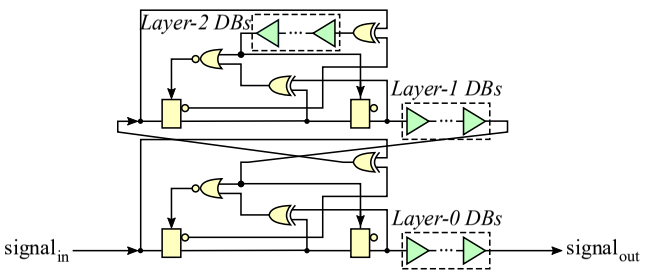
Since the logical operation on each layer inside of DPU-k is the same as that of DPU-1 containing one layer, I will discuss the logical correctness by using the circuit of DPU-1 in Figure 4.10(b). DPU-1 has two latches (marked as yellow rectangles), one placed next to the input signal $\text{signal}_{\text{in}}$ and the other next to the layer-0 delay buffers. Like the operation of my controller template explained in Section 4.2, $\text{signal}_{\text{in}}$ has to go through the layer-1 delay buffers first and sets en_{left} to 1 to pass the left latch. Meanwhile, the right latch blocks the output signal of the left latch from reaching



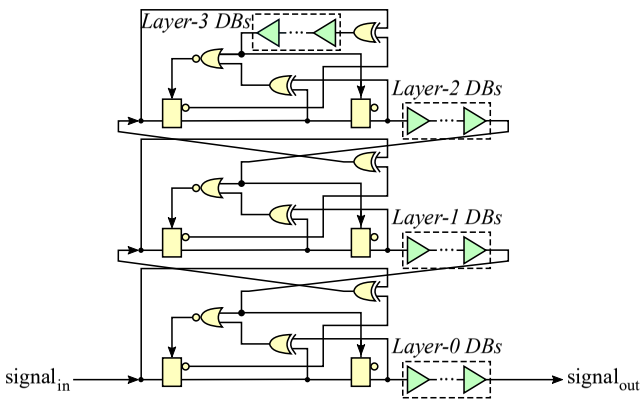
(a) DPU-0 (delay buffer chain)



(b) DPU-1



(c) DPU-2



(d) DPU-3

Figure 4.10: Four types of DPUs that are recursively to be applied to the modified circuit structure of the asynchronous pipeline controller template in Section 4.2. The synthesized timing path of the signal propagation is highlighted in red color, and the subsidiary logic cells are marked with yellow color.

to the layer-0 delay buffers until the latch enable signal en_{right} coming from the upper XOR through the layer-1 delay buffers becomes on. Note that the left latch is closed by resetting en_{left} to 0 through the lower XOR, regardless of the number of layer-1 delay buffers. If XOR_{lower} changed to 0 before the delayed XOR_{upper} is back to 1, en_{left} would become on once again, and as a result, the next $signal_{in}$ would pass through the left latch. However, en_{left} will never be triggered more than once by the single transition of $signal_{in}$ since one of the inputs to the lower XOR comes from the Q pin of the right latch.

Figure 4.11 shows the waveforms from the SPICE simulation on DPU-1 in Figure 4.10(b). First, the logic state of $signal_{in}$ (crimson wave) becomes 1, which triggers the transition of XOR_{upper} (pink wave) from 1 to 0. The output signal of the upper XOR, delayed by layer-1 delay buffers (purple wave), passes NOR and sets en_{left} (green wave) to 1, letting the left latch transparent. After passing the left latch, the inverted $signal_{in}$ resets XOR_{upper} to 1, and this transition goes through the layer-1 delay buffers once again and makes the right latch transparent by setting en_{right} from 0 to 1. As a result, $signal_{in}$ can pass the right latch and the layer-0 delay buffers. Figure 4.12 describes all of the possible logic behavior scenarios in DPU-1.

4.3.3 Updating Timing Constraints for Delay Path Units

Since the timing path of the input signal propagation will reuse the delay buffers in DPU, these signal transitions should be hazard-free to be logically correct operations, for which it has to satisfy the following constraint.

Constraint 3: Once the upper XOR on a layer in DPU has begun to accept the input signal transition of that layer, the signal to this XOR should unchange its state until the transition comes back to this XOR again through the NOR and the left latch on that layer.

Let $\Delta^{(l-1)}$ be the minimum holding time of the input signal to the l-th layer in DPU.

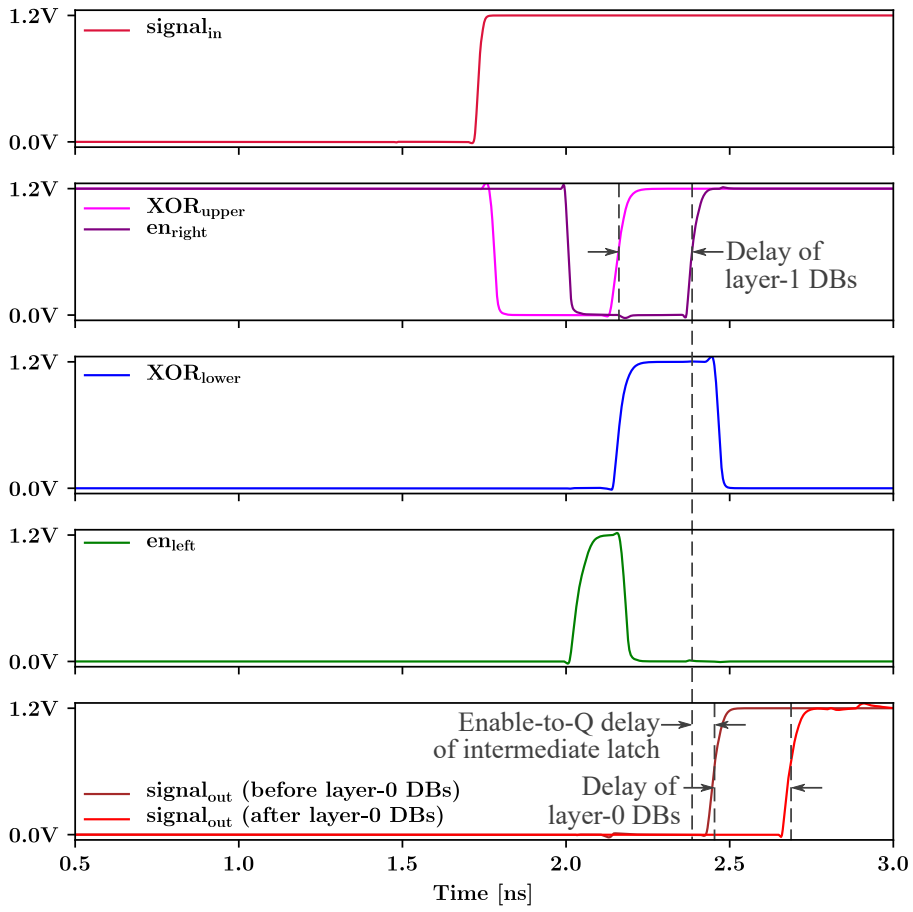


Figure 4.11: Simulation waveforms of DPU-1 in Figure 4.10(b).

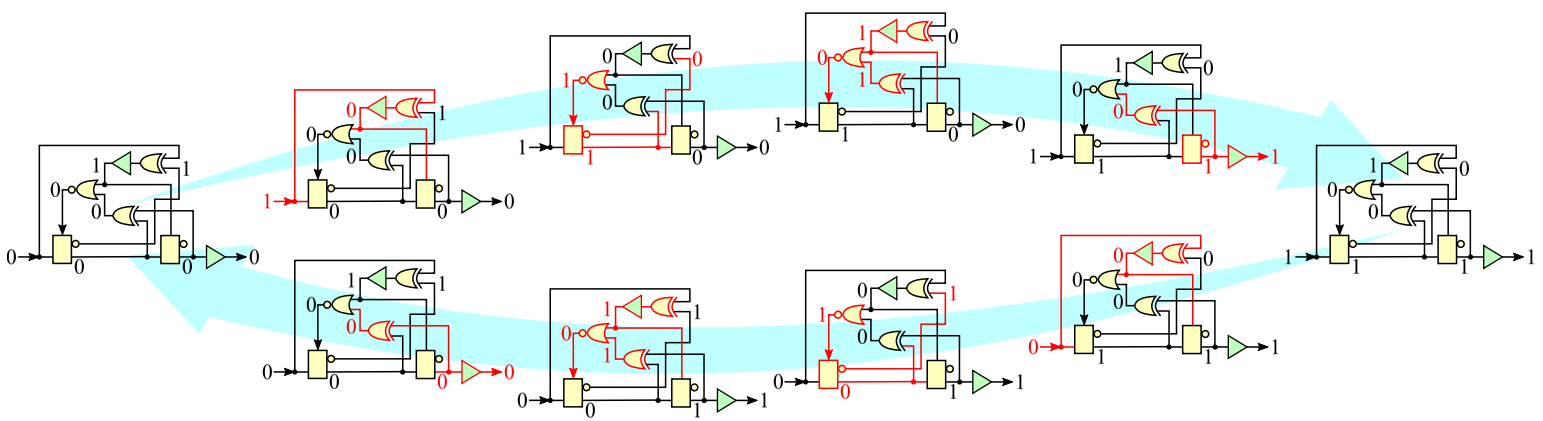


Figure 4.12: Logic behavior in DPU-1. The parts in red color indicate the change of logic values.

Then, DPU should satisfy the following inequality.

$$d^l_{\text{DB}} + d^l_{\text{NOR}} + d^{\text{left},l}_{\text{latch}} + d^l_{\text{XOR}} \leq \Delta^{(l-1)} \tag{4.7}$$

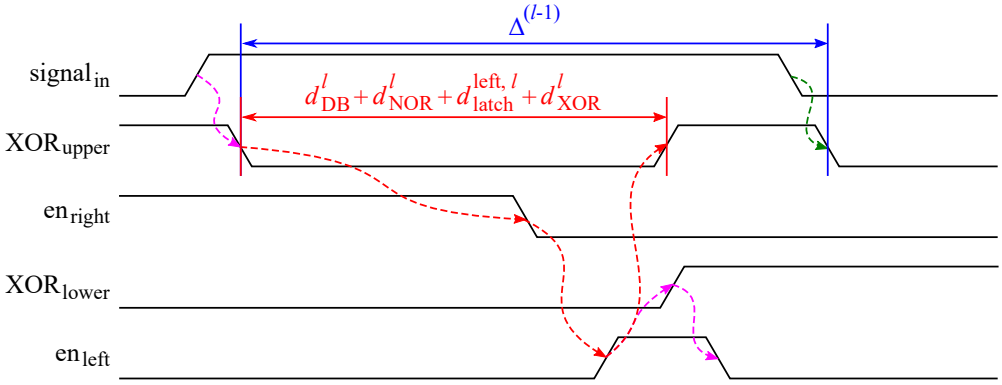
d^l_{DB} indicates the time interval during which the signal transition passes from the output of the upper XOR to the input of the NOR in the l-th layer in DPU in Figure 4.10(b). Figure 4.13 describes this constraint by waveforms and signal transition relation. In Figure 4.13(a), the transition of $\text{signal}_{\text{in}}$ passes the upper XOR, delay buffers, NOR, and then the left latch, setting $\text{XOR}_{\text{upper}}$ back to 1. In the meantime, the next $\text{signal}_{\text{in}}$ transition also repeats the same procedure, starting from the upper XOR. Thus $\text{XOR}_{\text{upper}}$ must be back to 1 before it is set to 0 by that transition. Figure 4.13(b) illustrates (temporal) non-conflicting two timing paths in a layer of DPU.

As explained previously, a DPU can replace either SDCs or NSDCs. The input signal to NSDC on pipeline stage i keeps its logic state during the cycle time of that pipeline stage, so that the minimum holding time for $l = 1$ will be $Cycle^i$ for DPU replacing that NSDC. On the other hand, the minimum holding time of the input signal to SDC on pipeline stage i is the smaller of the two time intervals during which the output of the upper XOR on that stage sustains 0-state and 1-state. Thus, I can calculate Δ^0 as

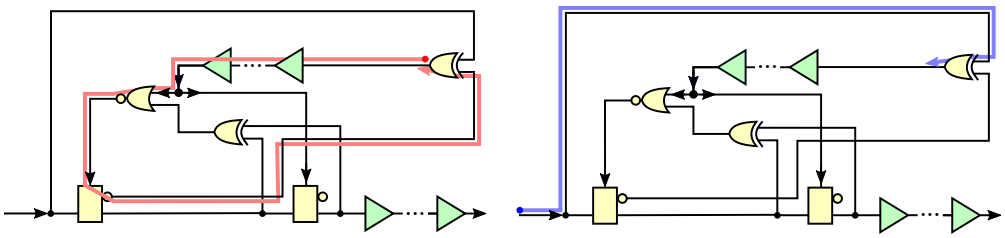
$$\begin{aligned}\Delta^0 = \min\{&d^i_{\text{SDC}} + d^i_{\text{NOR}} + d^i_{\text{latch}} + d^i_{\text{XOR}}, \\ &Cycle^{(i-1)} - (d^i_{\text{SDC}} + d^i_{\text{NOR}} + d^i_{\text{latch}} + d^i_{\text{XOR}})\}.\end{aligned}$$

Two terms in the *min*-function indicate the duration of sustaining 0-state and 1-state on output of the upper XOR in Figure 4.5, which corresponds to $\text{signal}_{\text{in}}$ of DPUs in Figure 4.10, and the minimum holding time of the input signal to the upper XOR on pipeline stage i (i.e., req^i) corresponds to the value of $Cycle^{(i-1)}$. Recursively, for the l-th layer in DPU, the minimum holding time can be expressed in terms of the minimum holding time of the lower $(l-1)$-th layer as

$$\begin{aligned}\Delta^l = \min\{&d^l_{\text{DB}} + d^l_{\text{NOR}} + d^{\text{left},l}_{\text{latch}} + d^l_{\text{XOR}}, \\ &\Delta^{(l-1)} - (d^l_{\text{DB}} + d^l_{\text{NOR}} + d^{\text{left},l}_{\text{latch}} + d^l_{\text{XOR}})\}.\end{aligned} \tag{4.8}$$



(a) Timing waveforms



(b) Two hazard-free timing paths

Figure 4.13: Timing waveforms and (temporal) non-conflicting timing path describing *Constraint 3*.

4.3.4 In-depth Synthesis Flow Utilizing Delay Path Units

While implementing asynchronous pipeline circuits using the LP formulation in Equation (4.6) in Section 4.2, it is not clear to apply the formulation directly to synthesizing my controller template with various types of DPUs. Hence, I propose a step-wise implementation procedure shown in Figure 4.14.

(Step 1: Preprocessing) For each pipeline stage, I first estimate the cycle time of the pipeline stage from its target latency computed from the timing analysis of its corresponding datapath after the completion of logic synthesis. Note that for each pipeline stage in my controller template, both of the cycle time and the latency of the pipeline stage include the same sharing and non-sharing delay circuits. Thus the cycle time of

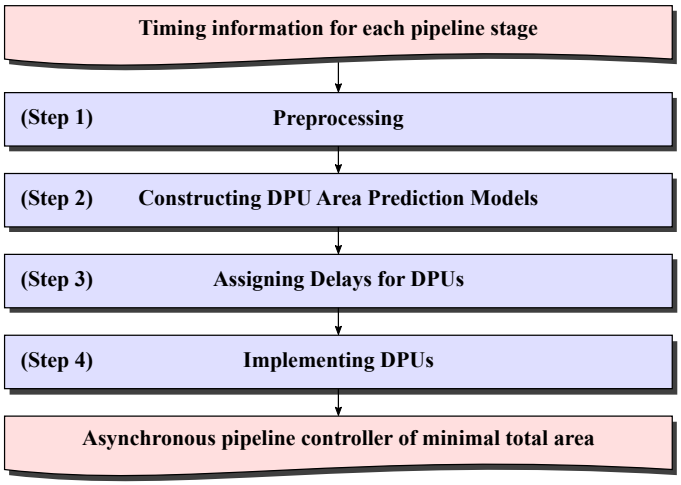


Figure 4.14: The flow of synthesizing an asynchronous pipeline controller.

the pipeline stage can be easily found out from the target latency before starting to allocate DPU.

(Step 2: Constructing DPU Area Prediction Models) Then, for each target propagation delay sample, I produce all feasible DPUs from which I build a piecewise linear model between the target delays and the area-minimal DPU implementation. Since the minimum holding times for SDCs and NSDCs are different, I construct the piecewise linear model for SDC as well as NSDC on every pipeline stage. Except for implementing the simple delay buffer chain (i.e., DPU-0), for synthesizing DPUs, all timing constraints in Section 4.3.3 should be met. Hence, for synthesizing DPU-L for a given minimum holding time Δ^0 and target delay d_{target}, I can compute the number of delay buffers minimally required in each layer in the DPU while satisfying all the constraints

by solving the following integer linear programming (ILP) formulation.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=0}^{L} n_l \\
\text{subject to} \quad & d_{\text{DB}}^{L} = n_L \times d_{\text{BUF}}, \\
& d_{\text{DB}}^{l} = d_0^1 + 2 \times d_{\text{DB}}^{(l+1)} + n_l \times d_{\text{BUF}}, \qquad \forall l \in \{0, \cdots, L-1\}, \\
& \Delta^l \leq d_{\text{DB}}^{l} + d_{\text{NOR}}^{l} + d_{\text{latch}}^{\text{left},l} + d_{\text{XOR}}^{l}, \qquad \forall l \in \{1, \cdots, L-1\}, \\
& \Delta^l \leq \Delta^{(l-1)} - (d_{\text{DB}}^{l} + d_{\text{NOR}}^{l} + d_{\text{latch}}^{\text{left},l} + d_{\text{XOR}}^{l}), \\
& \qquad\qquad \forall l \in \{1, \cdots, L-1\}, \\
& d_{\text{DB}}^{(l+1)} + d_{\text{NOR}}^{(l+1)} + d_{\text{latch}}^{\text{left},(l+1)} + d_{\text{XOR}}^{(l+1)} \leq \Delta^l, \\
& \qquad\qquad \forall l \in \{0, \cdots, L-1\}, \\
& d_{\text{target}} \leq d_{\text{DB}}^{0} \leq d_{\text{target}} + d_{\text{margin}}.
\end{aligned}
\tag{4.9}
$$

The first two constraints define the signal propagation delay from the output of the upper XOR to the input of the NOR for each layer in DPU-L except d_{DB}^0. The third and fourth constraints describe Equation (4.8) using two inequalities and restrict the minimum holding time of each layer in DPU-L. The fifth constraint corresponds to *Constraint 3* in Section 4.3.3, and the last one controls the difference between the target delay and my implementation. Note that d_0^1 and d_{margin} represent the propagation delay through DPU-1 with no insertion of delay buffers and the modeling margin, respectively. n_l indicates the number of the layer-l delay buffers.

(Step 3: Assigning Delays for DPUs) By referring to the piecewise linear models, I formulate the DPU allocation problem into an ILP:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in \mathcal{S}} \left(Area_S^i(d_{\text{SDC}}^i) + Area_{NS}^i(d_{\text{NSDC}}^i) \right) \\
\text{subject to} \quad & \text{Equations } (4.1), (4.2), (4.4), (4.5), && \forall i \in \mathcal{S}, \\
& d_{\text{latch}}^i + d_{\text{XOR}}^i + d_{\text{SDC}}^i + d_{\text{latch}}^{i \to (i+1)} \\
& \quad + d_{\text{NSDC}}^i + d_{\text{XOR}}^{(i+1)} + d_{\text{SDC}}^{(i+1)} + d_{\text{NOR}}^{(i+1)} \\
& \quad \leq Latency^i + \epsilon, && \forall i \in \mathcal{S}, \\
& d_{\text{latch}}^i + d_{\text{XOR}}^i + d_{\text{SDC}}^i + d_{\text{latch}}^{i \to (i+1)} \\
& \quad + d_{\text{NSDC}}^i + d_{\text{XOR}}^{(i+1)} + d_{\text{SDC}}^{(i+1)} + d_{\text{NOR}}^{(i+1)} \\
& \quad + d_{\text{latch}}^{(i+1)} + d_{\text{XOR}}^i + d_{\text{NOR}}^i \\
& \quad \leq Cycle^i + \epsilon, && \forall i \in \mathcal{S}.
\end{aligned}
\tag{4.10}
$$

$Area_S^i$ and $Area_{NS}^i$ are the mapping functions of the DPU of minimum-area on pipeline stage i for SDC and NSDC, respectively. Unlike Equation (4.6), the objective of Equation (4.10) includes the piecewise linear model, thus it becomes an ILP problem due to the inclusion of integer variables to indicate the selection among the intervals in the models. However, as will be seen from experiments, the time for solving this ILP formulation is not that significant since there are just a few intervals in the piecewise linear model.

(Step 4: Implementing DPUs) By solving Equation (4.9) once again with the delay values assigned to all the DPUs produced by the solution of Equation (4.10), I can complete the synthesis of the pipeline controller of minimal total area.

4.4 Experimental Results

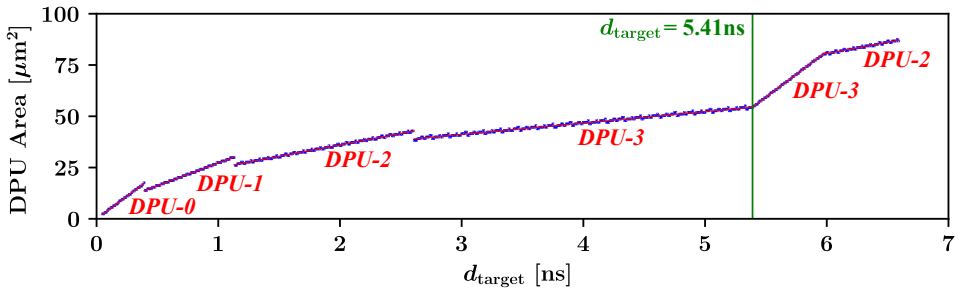
4.4.1 Environment Setup

Initially, I generated a set of pipelined circuits by serially linking the copies of ISCAS'85 combinational circuits for which I prepared two groups of pipelined architectures. The architectures in the first group were made by serially linking the same ISCAS'85 circuits with 4 (=3+1), 6 (=5+1), 11 (=10+1), 16 (=15+1), and 21 (=20+1) stages. For example, the circuit labeled C2670X5 has five copies of combinational circuit C2670, placed each one in between two consecutive pipeline stages. I generated the others in the same way, but their combinational circuits and the number of pipeline stages were chosen randomly among the ISCAS'85 designs and 4, 5, $\cdots$, 21, respectively. I implemented each combinational circuit using 45nm NanGate Open Cell Library [4] with Synopsys Design Compiler. Then, I extracted the maximum datapath delays between the pipeline stages using Synopsys PrimeTime for one specific corner for simplicity[2]. Note that the delay extraction can be easily extended to the case of multi-corner multi-mode scenarios by repeating the whole process. I used IBM CPLEX optimizer [113] through PICOS interface [114] for solving the LP formulation Equation (4.6) in Section 4.2.4 and ILP formulations Equations (4.9) and (4.10) in Section 4.3.4 with ϵ=10ps and d_{margin}=30ps. I also observed the timing behavior and measured the amounts of the dynamic and leakage power consumption of the controllers using Synopsys FineSim.

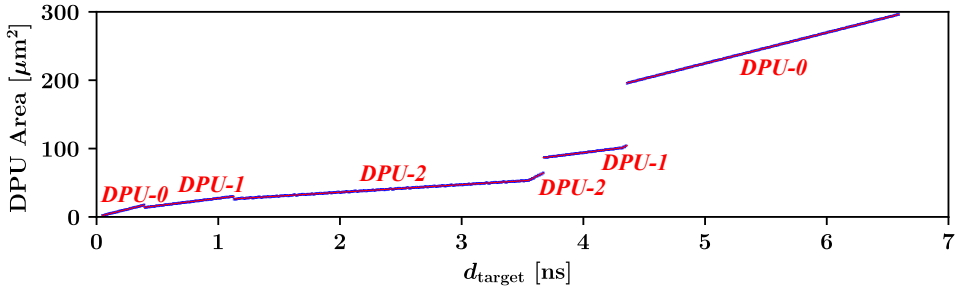
4.4.2 Piecewise Linear Modeling of Delay Path Unit Area

The red dots in Figure 4.15 show the minimum-area changes of DPU for SDC and NSDC corresponding to circuit C6288 as the target delay varies, derived by solving

[2]In the experiments, the temperature is set to $25\,^{\circ}\mathrm{C}$. However, as the temperature goes up to 70-$80\,^{\circ}\mathrm{C}$ while using deep sub-micron technology beyond 14nm, the leakage power consumption could significantly increase.



(a) DPU area for replacing NSDC (C6288)



(b) DPU area for replacing SDC (C6288)

Figure 4.15: Changes of the minimum implementation area for DPU allocation for NSDC and SDC corresponding to C6288 by varying the target delay. The blue dots and red lines represent the modeling samples and the piecewise linear prediction, respectively.

Equation (4.9) in Section 4.3.4. Figures 4.15(a) and (b) show the DPU allocation results for SDC and NSDC, for which in experiments I varied target delay d_{target} with a step size of 0.01ns. Note that for DPU replacing NSDC, the minimum holding time of input signal is decided by the cycle time of its pipeline stage, regardless of the delay generated by the DPU. On the other hand, for DPU replacing SDC, its holding time is limited by roughly half of the cycle time, and additional delay by the subsidiary cells makes this upper bound on minimum holding time tight. The blue lines in Figure 4.15 indicate the piecewise linear modeling results obtained from the samples.

I can observe that the slope of the DPU area increase becomes gradually gentle for small target delays in Figure 4.15. The reason is that I replaced the simple delay buffer chain (i.e., DPU-0) with DPUs with deep layers (e.g., DPU-1, DPU-2, DPU-3) that occupy less area. Meanwhile, even though the same type of DPU was used, the DPU area increases sharply for some target delays. Figure 4.16 illustrates the implementations for target delays of 5.38ns, 5.41ns, and 5.44ns in Figure 4.15(a), and as can be seen from the figure, I could substitute the less number of delay buffers in deeper layers for many delay buffers in the rest. However, as target delay exceeds some limit (e.g., 5.41ns in Figure 4.15(a)), delay buffer insertion in deeper layers violates *Constraint 3*, which causes not to maintain the increase rate of the DPU area as before.

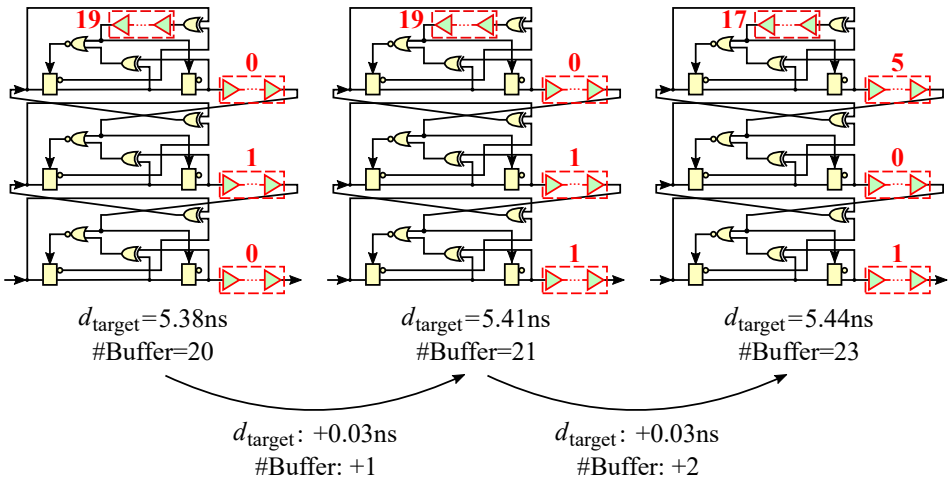


Figure 4.16: DPU implementations for target delays of 5.38ns, 5.41ns, and 5.44ns in Figure 4.15(a). The red values indicate the number of delay buffers.

Table 4.2 summarizes the runtime taken to prepare the data samples used in piecewise linear modeling for each combinational circuit in experiments. In the modeling, I used DPU-0 (i.e, a simple buffer chain), DPU-1, DPU-2, and DPU-3 in Figure 4.10 and set the step size of a target delay to 0.01ns. The preparation of all data samples was completed within two and a half minutes for every test case.

Table 4.2: The runtime of the data samples preparation used in the piecewise linear modeling step.

Circuit	C1908	C2670	C3540	C5315	C6288	C7552
Time [sec]	54	43	67	49	131	51

4.4.3 Comparison of Power, Performance, and Area

I notate DPSYN to refer to my controller synthesis technique in Section 4.2 that uses DPU-0. I also notate DPSYN+ to refer to my synthesis technique in Section 4.3 that uses DPU-1, DPU-2, DPU-3 as well as DPU-0. Table 4.3 summarizes the comparison of power, performance, and area of three implementations of asynchronous pipeline controllers: [3], DPSYN, and DPSYN+.

(Area) The column starts with 'Ckt.' in Table 4.3 is the name of pipeline circuits, and the number in each parenthesis represents the number of pipeline stages of the randomly generated circuit. Note that the numbers in parentheses in the second big column show the number of additional XORs and NORs for DPSYN+. The ratios of the total area of controllers (except logic gates shown in Figure 4.3) produced by DPSYN and DPSYN+ to that by [3] are shown in column 'Area,' from which DPSYN and DPSYN+ make 44.0% and 53.8% smaller controller on average over that by [3]. The runtimes to solve Equation (4.6) in DPSYN and to solve Equation (4.10) in DPSYN+ are less than 0.001 seconds and 0.382 seconds for all test cases.

(Performance) Columns 'Cycle Time' and 'Latency' indicate the average cycle time and latency normalized to their counterparts in [3], for all stages in each test case. The little increase in DPSYN and DPSYN+ is due to ensuring pessimism considering the gate delay models, but it can be controlled by adjusting ϵ in Equations (4.6) and (4.10) and d_{margin} in Equation (4.9). With the setting of ϵ=10ps and d_{margin}=30ps, the cycle time and latency of each pipeline stage increase only by 1.2% and 1.1% by DPSYN and 1.4% and 1.3% by DPSYN+ on average, in comparison with that by [3].

Table 4.3: Comparison of asynchronous pipeline controller implementations produced by [3], DPSYN, and DPSYN+ in terms of the number of additional logic gates, area, cycle time, latency, and leakage/dynamic power consumptions.

Ckt.	#Buffer+#Latch(+#XOR+#NOR)			Area		Cycle Time		Latency		Leakage Power		Dynamic Power	
	[3]	DPSYN	DPSYN+	DPSYN	DPSYN+	DPSYN	DPSYN+	DPSYN	DPSYN+	DPSYN	DPSYN+	DPSYN	DPSYN+
C1908X3	804+0	476+4	335+12(+8+4)	0.610	0.496	1.010	1.012	1.009	1.012	0.633	0.582	1.010	1.183
C1908X5	1122+0	630+6	422+18(+12+6)	0.581	0.462	1.010	1.013	1.010	1.013	0.607	0.558	1.011	1.217
C1908X10	1917+0	947+11	671+31(+20+10)	0.515	0.435	1.012	1.014	1.012	1.014	0.546	0.537	1.007	1.253
C1908X15	2712+0	1400+16	862+48(+32+16)	0.538	0.412	1.011	1.015	1.011	1.014	0.568	0.524	1.013	1.268
C1908X20	3507+0	1717+21	1106+61(+40+20)	0.512	0.408	1.012	1.015	1.011	1.014	0.544	0.518	1.008	1.275
C2670X3	627+0	368+4	275+10(+6+3)	0.610	0.521	1.005	1.013	1.004	1.013	0.639	0.598	1.019	1.112
C2670X5	875+0	486+6	396+12(+6+3)	0.581	0.520	1.005	1.015	1.004	1.014	0.613	0.591	1.020	1.086
C2670X10	1495+0	731+11	722+15(+4+2)	0.516	0.526	1.006	1.017	1.005	1.016	0.555	0.557	1.015	1.041
C2670X15	2115+0	1076+16	1001+22(+6+3)	0.536	0.519	1.005	1.017	1.004	1.016	0.574	0.580	1.022	1.050
C2670X20	2735+0	1321+21	1327+25(+4+2)	0.511	0.522	1.006	1.017	1.004	1.017	0.552	0.552	1.015	1.034
C3540X3	897+0	532+4	337+12(+8+4)	0.609	0.447	1.011	1.009	1.010	1.009	0.630	0.527	1.008	1.134
C3540X5	1253+0	705+6	414+18(+12+6)	0.580	0.407	1.011	1.009	1.010	1.008	0.603	0.497	1.009	1.148
C3540X10	2143+0	1060+11	589+33(+22+11)	0.513	0.357	1.012	1.008	1.012	1.007	0.542	0.459	1.009	1.163
C3540X15	3033+0	1570+16	799+48(+32+16)	0.537	0.348	1.012	1.008	1.011	1.007	0.564	0.454	1.010	1.169
C3540X20	3923+0	1925+21	974+63(+42+21)	0.510	0.334	1.012	1.008	1.011	1.007	0.539	0.442	1.009	1.172
C5315X3	714+0	422+4	342+8(+4+2)	0.612	0.534	1.010	1.013	1.009	1.012	0.637	0.562	1.013	1.056
C5315X5	996+0	480+6	480+10(+4+2)	0.582	0.529	1.010	1.014	1.010	1.014	0.611	0.556	1.014	1.046
C5315X10	1701+0	839+11	807+17(+6+3)	0.517	0.520	1.012	1.016	1.012	1.015	0.551	0.576	1.010	1.054
C5315X15	2406+0	1238+16	1193+20(+4+2)	0.539	0.530	1.011	1.016	1.010	1.016	0.572	0.582	1.016	1.036
C5315X20	3111+0	1519+21	1497+27(+6+3)	0.513	0.518	1.012	1.016	1.011	1.016	0.549	0.571	1.011	1.039
C6288X3	1829+0	1091+4	683+16(+12+6)	0.605	0.422	1.010	1.010	1.009	1.010	0.615	0.472	0.989	1.095
C6288X5	2557+0	1450+6	938+20(+14+7)	0.576	0.409	1.010	1.010	1.010	1.010	0.587	0.456	0.989	1.077

C6288X10	4377+0	2177+11	1533+31(+20+10)	0.507	0.388	1.012	1.010	1.012	1.009	0.521	0.434	0.984	1.061
C6288X15	6197+0	3245+16	2213+40(+24+12)	0.533	0.390	1.011	1.010	1.011	1.010	0.547	0.434	0.988	1.050
C6288X20	8017+0	3972+21	2808+51(+30+15)	0.505	0.383	1.012	1.010	1.012	1.010	0.520	0.427	0.985	1.048
C7552X3	728+0	431+4	354+10(+6+3)	0.612	0.557	1.011	1.012	1.010	1.011	0.637	0.632	1.014	1.121
C7552X5	1016+0	570+6	470+14(+8+4)	0.583	0.533	1.011	1.013	1.010	1.012	0.610	0.611	1.015	1.122
C7552X10	1736+0	857+11	715+25(+14+7)	0.517	0.485	1.013	1.014	1.012	1.013	0.551	0.569	1.010	1.123
C7552X15	2456+0	1265+16	1050+34(+18+9)	0.539	0.497	1.012	1.014	1.011	1.013	0.572	0.581	1.016	1.124
C7552X20	3176+0	1552+21	1295+45(+24+12)	0.513	0.479	1.012	1.014	1.012	1.014	0.548	0.565	1.011	1.124
RND1 (21)	3391+0	1897+21	1267+51(+30+15)	0.582	0.451	1.014	1.015	1.013	1.014	0.609	0.536	1.007	1.136
RND2 (8)	1209+0	608+8	533+18(+10+5)	0.527	0.516	1.015	1.015	1.014	1.015	0.562	0.591	1.003	1.099
RND3 (9)	1475+0	873+9	440+29(+20+10)	0.614	0.404	1.011	1.014	1.010	1.013	0.640	0.517	1.005	1.222
RND4 (10)	1425+0	800+10	592+20(+10+5)	0.587	0.484	1.010	1.014	1.010	1.013	0.618	0.565	1.008	1.104
RND5 (11)	1449+0	853+11	579+31(+20+10)	0.617	0.513	1.009	1.017	1.008	1.017	0.647	0.626	1.024	1.217
RND6 (17)	3307+0	1903+17	1216+47(+40+15)	0.594	0.442	1.010	1.014	1.010	1.013	0.617	0.526	1.006	1.151
RND7 (8)	1272+0	658+8	566+16(+8+4)	0.540	0.507	1.012	1.014	1.011	1.014	0.573	0.582	1.016	1.104
RND8 (20)	2858+0	1660+20	1056+50(+30+15)	0.606	0.460	1.011	1.015	1.010	1.015	0.635	0.561	1.002	1.165
RND9 (15)	2169+0	1147+15	1004+29(+14+7)	0.554	0.528	1.012	1.017	1.011	1.017	0.587	0.607	1.021	1.137
RND10 (13)	1918+0	986+13	773+27(+14+7)	0.539	0.473	1.012	1.016	1.011	1.015	0.573	0.557	1.010	1.110
RND11 (6)	900+0	538+6	294+18(+12+6)	0.622	0.433	1.011	1.016	1.010	1.016	0.650	0.546	1.003	1.179
RND12 (15)	2226+0	1193+15	840+35(+20+10)	0.561	0.457	1.013	1.014	1.012	1.014	0.592	0.550	1.004	1.117
RND13 (5)	773+0	400+5	322+11(+6+3)	0.541	0.488	1.013	1.014	1.013	1.013	0.575	0.571	1.001	1.114
RND14 (5)	1241+0	659+5	337+23(+18+9)	0.546	0.376	1.013	1.009	1.012	1.008	0.567	0.484	0.987	1.274
RND15 (16)	2685+0	1453+16	952+42(+26+13)	0.563	0.436	1.012	1.018	1.011	1.018	0.591	0.527	1.002	1.145
RND16 (7)	1298+0	762+7	416+27(+20+10)	0.607	0.435	1.012	1.013	1.012	1.012	0.630	0.544	1.002	1.241
RND17 (21)	3105+0	1836+21	1326+47(+26+13)	0.615	0.504	1.012	1.017	1.011	1.016	0.643	0.589	1.008	1.132
RND18 (8)	1314+0	766+8	438+24(+16+8)	0.605	0.431	1.014	1.019	1.014	1.019	0.631	0.532	1.006	1.203
RND19 (16)	2795+0	1581+16	946+52(+36+18)	0.587	0.439	1.010	1.012	1.009	1.012	0.612	0.544	1.003	1.210
RND20 (7)	1532+0	764+7	618+19(+12+6)	0.515	0.468	1.015	1.016	1.014	1.015	0.541	0.540	1.002	1.146

RND21 (8)	1488+0	803+8	596+20(+12+6)	0.559	0.470	1.013	1.014	1.012	1.013	0.586	0.543	0.999	1.101
RND22 (9)	1806+0	916+9	584+29(+20+10)	0.525	0.410	1.012	1.013	1.011	1.012	0.552	0.503	0.988	1.185
RND23 (20)	2706+0	1422+20	986+46(+26+13)	0.553	0.451	1.015	1.016	1.014	1.016	0.588	0.555	1.016	1.179
RND24 (21)	3634+0	1899+21	1432+47(+26+13)	0.544	0.459	1.013	1.016	1.012	1.015	0.572	0.538	0.999	1.138
RND25 (17)	3051+0	1711+17	1099+49(+32+16)	0.581	0.445	1.011	1.012	1.010	1.012	0.606	0.539	1.000	1.184
RND26 (15)	2143+0	1080+15	756+37(+22+11)	0.530	0.442	1.015	1.016	1.015	1.015	0.565	0.543	1.008	1.189
RND27 (4)	663+0	380+4	290+10(+6+3)	0.595	0.515	1.011	1.011	1.010	1.010	0.624	0.600	1.022	1.130
RND28 (10)	1614+0	842+10	718+18(+8+4)	0.544	0.498	1.016	1.017	1.016	1.016	0.576	0.567	1.007	1.101
RND29 (18)	3391+0	1902+18	1261+50(+32+16)	0.580	0.450	1.010	1.015	1.009	1.015	0.604	0.534	0.995	1.136
RND30 (13)	1834+0	915+13	649+33(+20+10)	0.525	0.447	1.015	1.014	1.011	1.013	0.588	0.543	1.007	1.137
Average	-	-	-	0.560	0.462	1.011	1.014	1.011	1.013	0.588	0.543	1.007	1.137

(Power) I can observe that DPSYN and DPSYN+ save the leakage power by 41.2% and 45.7%, which is mainly due to the reduced number of delay buffers. On the other hand, since the dynamic power depends on the amount of internal and external transitions regardless of the total cell count, the power consumption remains almost the same level for DPSYN. The slight increase is caused by one new latch inserted to each pipeline stage. However, for DPSYN+, the dynamic power increases by 13.7% from the insertion of multiple new latches to DPUs. Figure 4.17 clearly shows the trend that the dynamic power consumption increases as the latch cost invested per delay buffer increases.

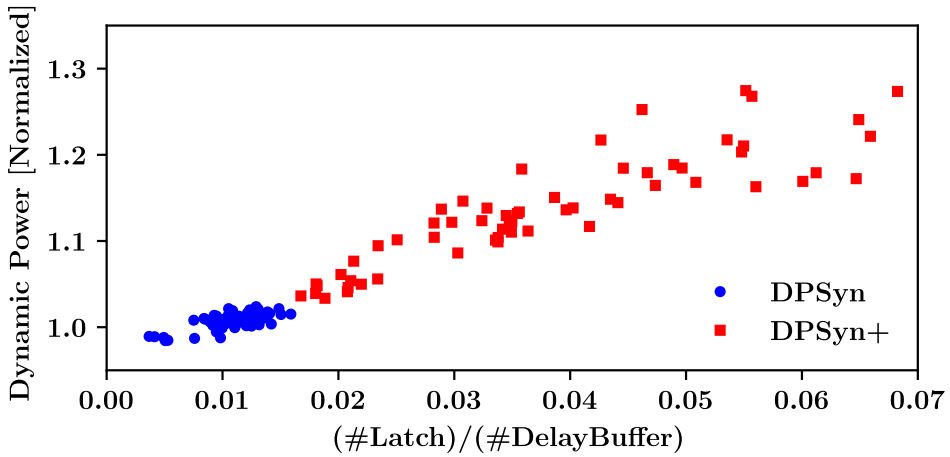
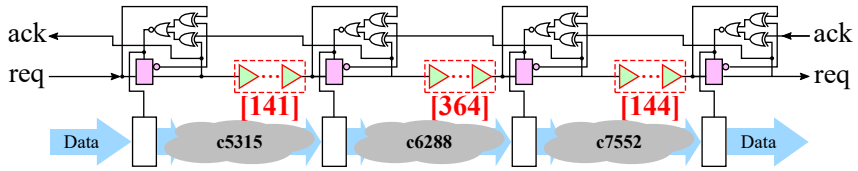
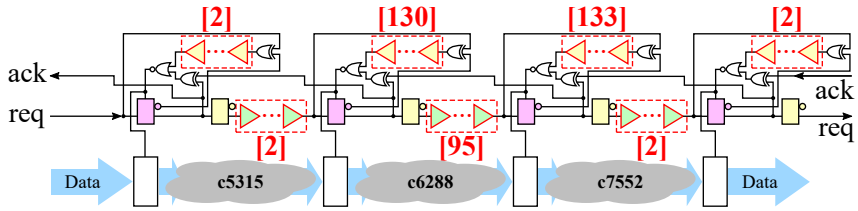


Figure 4.17: Changes in dynamic power consumption of the controller as the cost of extra latch per delay buffer increases. DPSYN uses a much lower cost than DPSYN+.

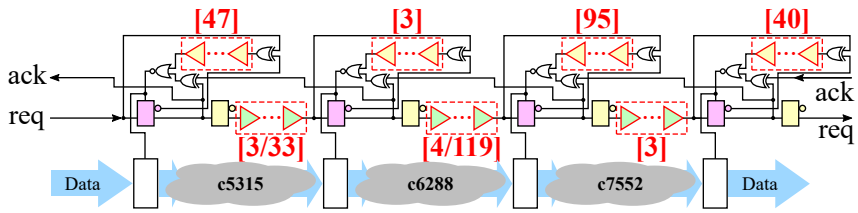
Figure 4.18 depicts the implementation results of the 3-stage pipelined circuits consisting of C5315, C6288, and C7552. DPSYN and DPSYN+ use 43.6% and 46.5% fewer number of delay buffers (from 649 to 366 and 347) at the expense of 4 more latches for DPSYN and 8 latches, 4 XORs, and 2 NORs for DPSYN+, respectively. Note that $[n_0/n_1]$ in Figure 4.18(c) indicates that the corresponding NSDC is replaced by DPU-1, in which n_0 and n_1 are the numbers of delay buffers inserted in the two



(a) Using [3] with DPU-0: 649 delay buffers



(b) Using my controller template with DPU-0: 366 delay buffers



(c) Using my controller template with DPU-0 and DPU-1: 347 delay buffers

Figure 4.18: The asynchronous pipeline controllers produced by (a) [3], (b) DPSYN, and (c) DPSYN+ for the 3-stage pipelined circuit consisting of C5315, C6288, and C7552. The red numbers indicate the number of delay buffers inserted into the layers.

layers. For example, $[3/33]$ indicates DPU-1 with 3 layer-0 delay buffers and 33 layer-1 delay buffers. $[n_0]$ (e.g., [47] in Figure 4.18(c)) represents the delay buffer chain consisting of n_0 delay buffers.

4.5 Summary

This chapter addressed the synthesis problem of two-phase bundled-data asynchronous pipeline controllers. To lighten the pipeline controllers, I developed a new logic synthesis concept called *delay path sharing and reusing*, by which I could significantly reduce the amount of the costly delay buffers. Precisely, (1) I proposed a technique

of synthesizing a pipeline controller in a way to share delay buffers among the setup timing paths for minimally allocating them; (2) I devised an area-efficient delay circuit structure called *delay path unit* (DPU) by extending the delay path sharing and proposed an in-depth synthesis flow of an asynchronous pipeline controller using DPUs. Through experiments, it was confirmed that my techniques of synthesizing asynchronous pipeline controllers were able to reduce the controller area by 44.0%∼53.8% and the leakage power by 41.2%∼45.7% on average while retaining the same level of performance.

Chapter 5

CONCLUSION

5.1 Chapter 2

In Chapter 2, I pointed out the limitations of the pointwise manner used in the conventional approaches related to flexible flip-flop timing model based STA through two examples. To complement them, I proposed the concept of clock-to-Q delay interval analysis and introduced two additional timing constraints for this. By applying it, I formulated the representative timing analysis problems (i.e., finding the minimum clock period of a given circuit and clock skew scheduling for maximizing the worst and total timing slacks) into the instances of convex optimization. In addition, I suggested a pre-processing algorithm for ameliorating the speed and scalability of solving the problems. Through the experiments with a 45nm cell library, I demonstrated the discrepancy between the results of previous and proposed timing analysis approaches and improved the quality of timing analysis and optimization for the problems in terms of the clock period, worst timing slack, and total timing slack. Furthermore, my scalable speedup technique reduced the runtimes of solving significantly at the expense of little loss of optimality and enabled the handling of large-size problem instances as well.

5.2 Chapter 3

In Chapter 3, to deal with the performance variation in an NTV regime and advanced process technology, I proposed a holistic HPM methodology throughout typical IC design flow from design to post-silicon phase. Specifically, I first formulated the problem of finding an efficient configuration of monitoring circuits into the instance of the optimal experiment design problem. Besides, I suggested a new target timing prediction flow combining statistical estimation of FEOL and BEOL PVs and neural network based timing inference model with accurate timing margin control via uncertainty learning. To resolve simulation-silicon discrepancies, I built a prediction model calibration flow in a silicon characterization step with transfer learning. I also shortened the preparation time of the prediction model by substituting efficient but accurate surrogate models for time-consuming SPICE simulations. The experimental results with a 28nm industry PDK and DK characterized at 0.6V demonstrated that the average prediction pessimism of maximum delay reduced by 77.9% over the conventional signoff results while respecting target prediction yield. Furthermore, for test chips fabricated using a 10nm process in conjunction with AVS, I saved dynamic power consumption by 28.2%$\sim$28.8% on average.

5.3 Chapter 4

Lightening a pipeline controller directly impacts two domains: mitigating the increase of controller area by PVT variation and reducing leakage power consumption. In Chapter 4, I targeted employing a new delay circuit structure as well as resynthesizing the conventional state-of-the-art controller to achieve the two factors in the domains for two-phase bundled-data asynchronous pipeline controllers while retaining all the previous benefits. First, I reduced the number of delay buffers ensuring the correctness of timing behavior significantly by modifying the setup timing paths in a pipeline controller. In addition, I devised a new area-efficient delay circuit structure called de-

lay path unit (DPU) through delay path reusing, which is the extension of delay path sharing, and suggested in-depth synthesis flow of an asynchronous pipeline controller using it. Through the experiments with a 45nm cell library, I validated the efficacy of my proposed techniques. Precisely, over the conventional asynchronous pipeline controller, I reduced the area and leakage power by 44.0%∼53.8% and 41.2%∼45.7% on average, respectively.

Bibliography

[1] J. P. Fishburn, "Clock skew optimization," *IEEE Transactions on Computers*, vol. 39, no. 7, pp. 945–951, 1990.

[2] H. Seo, J. Heo, and T. Kim, "Post-silicon tuning based on flexible flip-flop timing," vol. 16, no. 1, pp. 11–22, 2016.

[3] K.-H. Ho and Y.-W. Chang, "A new asynchronous pipeline template for power and performance optimization," in *Design Automation Conference*, 2014, pp. 1–6.

[4] "Nangate 45nm open cell library," http://www.nangate.com, 2011.

[5] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[6] "International roadmap for device and systems," https://irds.ieee.org, 2017.

[7] C. Meinhardt, A. L. Zimpeck, and R. A. L. Reis, "Predictive evaluation of electrical characteristics of sub-22 nm finfet technologies under device geometry variations," *Microelectronics Reliability*, vol. 54, no. 9-10, pp. 2319–2324, 2014.

[8] A. Kapoor, N. Jayakumar, and S. P. Khatri, "A novel clock distribution and dynamic de-skewing methodology," in *International Conference on Computer-Aided Design, 2004*, 2004, pp. 626–631.

[9] J. Myers, A. Savanth, P. Prabhat, S. Yang, R. Gaddh, S. O. Toh, and D. Flynn, "A 12.4pj/cycle sub-threshold, 16pj/cycle near-threshold arm cortex-m0+ mcu with autonomous srpg/dvfs and temperature tracking clocks," in *Symposium on VLSI Circuits*, 2017, pp. C332–C333.

[10] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" in *Advances in Neural Information Processing Systems*, 2017, pp. 5574–5584.

[11] V. Stojanovic and V. G. Oklobdzija, "Comparative analysis of master-slave latches and flip-flops for high-performance and low-power systems," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 4, pp. 536–548, 1999.

[12] A. M. Jain and D. Blaauw, "Modeling flip flop delay dependencies in timing analysis," in *Timing Issues (TAU) Workshops*, 2004, pp. 67–73.

[13] G. G. Rao and E. K. Howick Jr., "Apparatus for optimized constraint characterization with degradation options and associated methods," Jun. 24 2003, US Patent 6,584,598.

[14] S. Srivastava and J. Roychowdhury, "Interdependent latch setup/hold time characterization via euler-newton curve tracing on state-transition equations," in *Design Automation Conference*, 2007, pp. 136–141.

[15] S. Srivastava and J. Roychowdhury, "Independent and interdependent latch setup/hold time characterization via newton-raphson solution and eular curve tracking of state-transition equations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 5, pp. 817–830, 2008.

[16] E. Salman, A. Dasdan, F. Taraporevala, K. Kucukcakar, and E. G. Friedman, "Pessimism reduction in static timing analysis using interdependent setup and hold times," in *International Symposium on Quality Electronic Design*, 2006.

[17] E. Salman, A. Dasdan, F. Taraporevala, K. Kucukcakar, and E. G. Friedman, "Exploiting setup–hold-time interdependence in static timing analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 6, pp. 1114–1125, 2007.

[18] E. Salman and E. G. Friedman, "Utilizing interdependent timing constraints to enhance robustness in synchronous circuits," *Microelectronics Journal*, vol. 43, no. 2, pp. 119–127, 2012.

[19] S. Hatami, H. Abrishami, and M. Pedram, "Statistical timing analysis of flip-flops considering codependent setup and hold times," in *Great Lakes Symposium on VLSI*, 2008, pp. 101–106.

[20] N. Chen, B. Li, and U. Schlichtmann, "Iterative timing analysis based on non-linear and interdependent flipflop modelling," *IET Circuits, Devices & Systems*, vol. 6, no. 5, pp. 330–337, 2012.

[21] A. B. Kahng and H. Lee, "Timing margin recovery with flexible flip-flop timing model," in *International Symposium on Quality Electronic Design*, 2014, pp. 496–503.

[22] H. Seo, J. Heo, and T. Kim, "Clock skew optimization for maximizing time margin by utilizing flexible flip-flop timing," in *International Symposium on Quality Electronic Design*, 2015, pp. 35–39.

[23] Y.-M. Yang, K. H. Tam, and I. H.-R. Jiang, "Criticality-dependency-aware timing characterization and analysis," in *Design Automation Conference*, 2015.

[24] M. B. Taylor, "Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse," in *Design Automation Conference*, 2012, pp. 1131–1136.

[25] J. Henkel, H. Khdr, S. Pagani, and M. Shafique, "New trends in dark silicon," in *Design Automation Conference*, 2015, pp. 1–6.

[26] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[27] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Design Automation Conference*, 2003, pp. 338–342.

[28] K. J. Kuhn, M. D. Giles, D. Becher, P. Kolar, A. Kornfeld, R. Kotlyar, S. T. Ma, A. Maheshwari, and S. Mudanai, "Process technology variation," *IEEE Transactions on Electron Devices*, vol. 58, no. 8, pp. 2197–2208, 2011.

[29] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer, "Statistical timing analysis: From basic principles to state of the art," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 589–607, 2008.

[30] M. Bhushan, A. Gattiker, M. B. Ketchen, and K. K. Das, "Ring oscillators for cmos process tuning and variability control," *IEEE Transactions on Semiconductor Manufacturing*, vol. 19, no. 1, pp. 10–18, 2006.

[31] I. A. K. M. Mahfuzul, A. Tsuchiya, K. Kobayashi, and H. Onodera, "Variation-sensitive monitor circuits for estimation of global process parameter variation,"

IEEE Transactions on Semiconductor Manufacturing, vol. 25, no. 4, pp. 571–580, 2012.

[32] K. Kang, S. P. Park, K. Kim, and K. Roy, "On-chip variability sensor using phase-locked loop for detecting and correcting parametric timing failures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 2, pp. 270–280, 2009.

[33] A. J. Drake, R. M. Senger, H. Singh, G. D. Carpenter, and N. K. James, "Dynamic measurement of critical-path timing," in *International Conference on Integrated Circuit Design and Technology and Tutorial*, 2008, pp. 249–252.

[34] T.-B. Chan and A. B. Kahng, "Tunable sensors for process-aware voltage scaling," in *International Conference of Computer-Aided Design*, 2012, pp. 7–14.

[35] L. M. Burns, L. Dauphinee, R. A. Gomez, and J. Y. C. Chang, "Process monitor for monitoring and compensating circuit performance," May 20 2008, US Patent 7,375,540.

[36] T.-B. Chan, A. Pant, L. Cheng, and P. Gupta, "Design dependent process monitoring for back-end manufacturing cost reduction," in *International Conference on Computer-Aided Design*, 2010, pp. 116–122.

[37] D. J. Pilling, "In-situ monitor of process and device parameters in integrated circuits," Sep. 22 2009, US Patent 7,594,149.

[38] T. A. Black, "A critical path based parametric ring oscillator," Master's thesis, Texas Tech University, 2000.

[39] D. Fick, N. Liu, Z. Foo, M. Fojtik, J. sun Seo, D. Sylvester, and D. Blaauw, "In situ delay-slack monitor for high-performance processors using an all-digital self-calibrating 5ps resolution time-to-digital converter," in *International Solid-State Circuits Conference*, 2010, pp. 188–189.

[40] L. Lai, V. Chandra, R. Aitken, and P. Gupta, "Slackprobe: A low overhead in situ on-line timing slack monitoring methodology," in *Design, Automation & Test in Europe Conference & Exhibition*, 2013, pp. 282–287.

[41] H. C. Ngo, G. D. Carpenter, A. J. Drake, and J. B. Kuang, "Circuit timing monitor having a selectable-path ring oscillator," Oct. 5 2010, US Patent 7,810,000.

[42] K. Shaik, "Implementation of a critical path based parametric ring oscillator," Master's thesis, Texas Tech University, 2001.

[43] X. Wang, M. Tehranipoor, S. George, D. Tran, and L. Winemberg, "Design and analysis of a delay sensor applicable to process/environmental variations and aging measurements," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 8, pp. 1405–1418, 2012.

[44] L. Xie and A. Davoodi, "Representative path selection for post-silicon timing prediction under variability," in *Design Automation Conference*, 2010, pp. 386–391.

[45] Q. Liu and S. S. Sapatnekar, "A framework for scalable postsilicon statistical delay prediction under process variations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 8, pp. 1201–1212, 2009.

[46] Q. Liu and S. S. Sapatnekar, "Capturing post-silicon variations using a representative critical path," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 2, pp. 211–222, 2010.

[47] T.-B. Chan, A. B. Kahng, and J. Li, "Nolo: A no-loop, predictive useful skew methodology for improved timing in ic implementation," in *International Symposium on Quality Electronic Design*, 2014, pp. 504–509.

[48] S.-P. Mu, M. C.-T. Chao, S.-H. Chen, and Y.-M. Wang, "Statistical framework and built-in self-speed-binning system for speed-binning using on-chip ring oscillators," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 5, pp. 1675–1687, 2016.

[49] M. Sadi, S. Kannan, L. Winemberg, and M. Tehranipoor, "Soc speed binning using machine learning and on-chip slack sensors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 5, pp. 842–854, 2017.

[50] W. Dai and D. Staepelaere, "Useful-skew clock tree synthesis boosts asic performance," in *Closing the gap between asic & custom*, D. Chinnery and K. Keutzer, Eds. Springer, 2004, pp. 209–223.

[51] H. Xu, V. F. Pavlidis, W. Burleson, and G. D. Micheli, "The combined effect of process variations and power supply noise on clock skew and jitter," in *International Symposium on Quality Electronic Design*, 2012, pp. 320–327.

[52] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low power methodology manual: For system-on-chip design*. Springer, 2007.

[53] J. Rabaey, *Low power design essentials*. Springer, 2009.

[54] G. Frantz, "Giving technology 2020 vision," TI presentation, 2008.

[55] P. A. Beerel, R. O. Ozdag, and M. Ferretti, *A designer's guide to asynchronous VLSI*. Cambridge University Press, 2010.

[56] N. C. Paver, "The design and implementation of an asynchronous microprocessor," Ph.D. dissertation, University of Manchester, 1994.

[57] A. J. Martin, M. Nyström, and C. G. Wong, "Three generations of asynchronous microprocessors," *IEEE Design & Test of Computers*, vol. 20, no. 6, pp. 9–17, 2003.

[58] A. J. Martin, A. Lines, R. Manohar, M. Nyström, P. Penzes, R. Southworth, U. Cummings, and T. K. Lee, "The design of an asynchronous mips r3000 microprocessor," in *Conference on Advanced Research in VLSI*, 1997, pp. 164–181.

[59] S. B. Furber, J. D. Garside, N. C. Paver, and J. V. Woods, "Amulet1: a micropipelined arm," in *Proceedings of COMPCON'94*, 1994, pp. 476–485.

[60] S. B. Furber, J. D. Garside, P. Riocreux, S. Temple, P. Day, J. Liu, and N. C. Paver, "Amulet2e: An asynchronous embedded controller," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 243–256, 1999.

[61] J. D. Garside, W. J. Bainbridge, A. Bardsley, D. M. Clark, D. A. Edwards, S. B. Furber, J. Liu, D. W. Lloyd, S. Mohammadi, J. S. Pepper, O. Petlin, S. Temple, and J. V. Woods, "Amulet3i-an asynchronous system-on-chip," in *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 2000, pp. 162–175.

[62] T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako, and A. Takamura, "Titac: Design of a quasi-delay-insensitive microprocessor," *IEEE Design & Test of Computers*, vol. 11, no. 2, pp. 50–63, 1994.

[63] A. Takamura, M. Kuwako, M. Imai, T. Fujii, M. Ozawa, I. Fukasaku, Y. Ueno, and T. Nanya, "Titac-2: An asynchronous 32-bit microprocessor based on scalable-delay-insensitive model," in *International Conference on Computer Design VLSI in Computers and Processors*, 1997, pp. 288–294.

[64] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann, "An asynchronous low-power 80c51 microcontroller," in *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1998, pp. 96–107.

[65] S. M. Nowick and M. Singh, "Asynchronous design-part 2: Systems and methodologies," *IEEE Design & Test*, vol. 32, no. 3, pp. 19–28, 2015.

[66] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[67] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[68] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2012.

[69] M. Ferretti and P. A. Beerel, "High performance asynchronous design using single-track full-buffer standard cells," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 6, pp. 1444–1454, 2006.

[70] D. Hand, M. T. Moreira, H.-H. Huang, D. Chen, F. Butzke, Z. Li, M. Gibiluka, M. Breuer, N. L. V. Calazans, and P. A. Beerel, "Blade–a timing violation resilient asynchronous template," in *International Symposium on Asynchronous Circuits and Systems*, 2015, pp. 21–28.

[71] J. Simatic, A. Cherkaoui, F. Bertrand, R. P. Bastos, and L. Fesquet, "A practical framework for specification, verification, and design of self-timed pipelines," in *International Symposium on Asynchronous Circuits and Systems*, 2017, pp. 65–72.

[72] S. M. Nowick and M. Singh, "High-performance asynchronous pipelines: An overview," *IEEE Design & Test of Computers*, vol. 28, no. 5, pp. 8–22, 2011.

[73] I. Sutherland and S. Fairbanks, "Gasp: A minimal fifo control," in *International Symposium on Asynchronous Circuits and Systems*, 2001, pp. 46–53.

[74] M. Singh and S. M. Nowick, "The design of high-performance dynamic asynchronous pipelines: High-capacity style," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 11, pp. 1270–1283, 2007.

[75] K. M. Fant and S. A. Brandt, "Null convention logic^{TM}: A complete and consistent logic for asynchronous digital circuit synthesis," in *International Conference of Application Specific Systems, Architectures and Processors*, 1996, pp. 261–273.

[76] A. J. Martin and M. Nyström, "Asynchronous techniques for system-on-chip design," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1089–1120, 2006.

[77] Z. Xia, S. Ishihara, M. Hariyama, and M. Kameyama, "Dual-rail/single-rail hybrid logic design for high-performance asynchronous circuit," in *International Symposium on Circuits and Systems*, 2012, pp. 3017–3020.

[78] A. Bardsley, "Balsa: An asynchronous circuit synthesis system," Ph.D. dissertation, University of Manchester, 1998.

[79] R. B. Reese, S. C. Smith, and M. A. Thornton, "Uncle-an rtl approach to asynchronous design," in *International Symposium on Asynchronous Circuits and Systems*, 2012, pp. 65–72.

[80] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, 1989.

[81] M. Singh and S. M. Nowick, "Mousetrap: High-speed transition-signaling asynchronous pipelines," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 6, pp. 684–698, 2007.

[82] N. V. Toan, D. M. Tung, and J.-G. Lee, "Energy-efficient and high performance 2-phase asynchronous micropipelines," in *International Midwest Symposium on Circuits and Systems*, 2017, pp. 1188–1191.

[83] O. L. Mangasarian, J. B. Rosen, and M. E. Thompson, "Global minimization via piecewise-linear underestimation," *Journal of Global Optimization*, vol. 32, no. 1, pp. 1–9, 2005.

[84] A. Magnani and S. P. Boyd, "Convex piecewise-linear fitting," *Optimization and Engineering*, vol. 10, no. 1, pp. 1–17, 2009.

[85] E. Lim and P. W. Glynn, "Consistency of multidimensional convex regression," *Operations Research*, vol. 60, no. 1, pp. 196–208, 2012.

[86] L. A. Hannah and D. B. Dunson, "Multivariate convex regression with adaptive partitioning," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 3261–3294, 2013.

[87] Y. Nesterov and A. Nemirovskii, *Interior-point polynomial algorithms in convex programming*. SIAM, 1994, vol. 13.

[88] "Cvx: Matlab software for disciplined convex programming, version 2.0," http://cvxr.com/cvx, August 2012.

[89] M. C. Grant and S. P. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent advances in learning and control*. Springer, 2008, pp. 95–110.

[90] K.-C. Toh, M. J. Todd, and R. H. Tütüncü, "Sdpt3-a matlab software package for semidefinite programming, version 1.3," *Optimization Methods and Software*, vol. 11, pp. 545–581, 1999.

[91] R. H. Tütüncü, K.-C. Toh, and M. J. Todd, "Solving semidefinite-quadratic-linear programs using sdpt3," *Mathematical Programming*, vol. 95, no. 2, pp. 189–217, 2003.

[92] T. Huynh-Bao, J. R. Z. Tőkei, A. Mercha, D. Verkest, A. V.-Y. Thean, and P. Wambacq, "Statistical timing analysis considering device and interconnect variability for beol requirements in the 5-nm node and beyond," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1669–1680, 2017.

[93] N. Lu and J. H. McCullen, "Enablement of variatio-aware timing: Treatment of parasitic resistance and capacitance," in *International Symposium on Quality Electronic Design*, 2007, pp. 743–748.

[94] K. Chiang, J.-F. Huang, T.-Y. Cheng, C. Hsiao, J. Sun, C. Cheng, K.-P. Lu, K.-W. Su, C.-K. Lin, K. Chen, K.-H. Tam, T.-Y. Liu, K.-Y. Su, and M.-C. Jeng, "A comprehensive solution for beol variation characterization and modeling," in *International Conference on Simulation of Semiconductor Processes and Devices*, 2016, pp. 307–310.

[95] R. G. Ghanem and P. D. Spanos, *Stochastic finite elements: a spectral approach*. Springer-Verlag New York Inc., 1991.

[96] Z. Zhang, T. A. El-Moselhy, I. M. Elfadel, and L. Daniel, "Stochastic testing method for transistor-level uncertainty quantiication based on generalized polynomial chaos," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 10, pp. 1533–1545, 2013.

[97] D. Xiu and G. E. Karniadakis, "The wiener–askey polynomial chaos for stochastic differential equations," *SIAM Journal on Scientific Computing*, vol. 24, no. 2, pp. 619–644, 2002.

[98] D. Xiu and J. S. Hesthaven, "High-order collection methods for differential equations with random inputs," *SIAM Journal on Scientific Computing*, vol. 27, no. 3, pp. 1118–1139, 2005.

[99] Z. Zhang, T.-W. Weng, and L. Daniel, "Big-data tensor recovery for high-dimensional uncertainty quantification of process varitions," *IEEE Transactions on Components, Packaging, and Manufacturing Technology*, vol. 7, no. 5, pp. 687–697, 2017.

[100] J. Chung and J. Kim, "Segment delay learning from quantized path delay measurements," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 1038–1042, 2015.

[101] V. V. Fedorov, *Theory of optimal experiments*. Academic Press, 1972.

[102] G. Sagnol and R. Harman, "Computing exact d-optimal designs by mixed integer second-order cone programming," *The Annals of Statistics*, vol. 43, no. 5, pp. 2198–2224, 2015.

[103] R. Harman, A. Bachratá, and L. Filová, "Construction of efficient experimental designs under multiple resource constraints," *Applied Stochastic Models in Business and Industry*, vol. 32, no. 1, pp. 3–17, 2016.

[104] P. Goos and B. Jones, *Optimal design of experiments: a case study approach*. John Wiley & Sons, 2011.

[105] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, University of Cambridge, 2016.

[106] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Advances in Neural Information Processing Systems*, 2017, pp. 6402–6413.

[107] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[108] J. Lu, V. Behbood, P. Hao, H. Zuo, S. Xue, and G. Zhang, "Transfer learning using computational intelligence: A survey," *Knowledge-Based Systems*, vol. 80, pp. 14–23, 2015.

[109] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems*, 2014, pp. 3320–3328.

[110] S. Kang, "On effectiveness of transfer learning approach for neural network-based virtual metrology modeling," *IEEE Transactions on Semiconductor Manufacturing*, vol. 31, no. 1, pp. 149–165, 2017.

[111] Y. Lin, M. Li, Y. Watanabe, T. Kimura, T. Matsunawa, S. Nojima, and D. Z. Pan, "Data efficienct lithography modeling with transfer learning and active data selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 10, pp. 1900–1913, 2019.

[112] G. Gimenez, A. Cherkaoui, G. Cogniard, and L. Fesquet, "Static timing analysis of asynchronous bundled-data circuits," in *International Symposium on Asynchronous Circuits and Systems*, 2018, pp. 110–118.

[113] "Ibm ilog cplex optimizer 12.8.0," https://www.ibm.com/analytics/cplex-optimizer, 2017.

[114] G. Sagnol, "Picos, a python interface to conic optimization solvers," Technical Report 12-48, ZIB. http://picos.zib.de, Tech. Rep., 2012.

초 록

타이밍 분석은 반도체 회로 개발 필수 과정 중 하나로, 최신 공정일수록 공정-전압-온도 변이 증가를 포함한 다양한 요인으로 하여금 그 중요성이 커지고 있다. 본 논문에서는 로직 및 피지컬 합성과 관련하여 세 가지 타이밍 분석 및 최적화 문제에 대해 다룬다. 첫째로, 오늘날 대부분의 정적 타이밍 분석은 모든 플립-플롭의 클럭-출력 딜레이가 고정된 값이라는 가정을 바탕으로 이루어졌다. 하지만 실제 클럭-출력 딜레이는 해당 플립-플롭의 셋업 및 홀드 스큐에 영향을 받는다. 본 논문에서는 이러한 특성을 수학적으로 정리하였으며, 이를 확장 가능한 속도 향상 기법과 더불어 주어진 회로의 타이밍 분석 및 클럭 스큐 스케쥴링 문제에 적용하였다. 둘째로, 유사 문턱 연산은 초고집적 회로 동작의 에너지 효율을 끌어 올릴 수 있다는 점에서 각광받지만, 큰 폭의 성능 변이 및 비선형성 때문에 널리 활용되고 있지 않다. 이를 해결하기 위해 유사 문턱 전압 영역 및 최신 공정 노드에서 보다 정확한 타이밍 예측을 위한 하드웨어 성능 모니터링 방법론 전반을 제안하였다. 마지막으로, 비동기 회로는 기존 동기 회로의 대안 중 하나로, 그 중에서도 비동기 파이프라인 회로는 비교적 적은 설계 노력만으로도 구현 가능하다는 장점이 있다. 본 논문에서는 2위상 묶음 데이터 프로토콜 기반 비동기 파이프라인 컨트롤러 상에서, 정확한 핸드셰이킹 통신을 위해 삽입된 딜레이 버퍼에 의한 면적 증가를 완화할 수 있는 합성 기법을 제시하였다.

주요어: 타이밍 분석, 플립-플롭, 하드웨어 성능 모니터링 방법론, 유사 문턱 연산, 비동기 회로, 파이프라인 컨트롤러, 딜레이 경로.
학번: 2014-21722